VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATIZOVANÉ TESTOVÁNÍ WEBOVÝCH APLIKACÍ

AUTOMATED OF WEB APPLICATION TESTING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. LUKÁŠ KÖSZEGY**
AUTHOR

VEDOUCÍ PRÁCE **doc. RNDr. JITKA KRESLÍKOVÁ, CSc.**
SUPERVISOR

BRNO 2018

Zadání diplomové práce/21246/2017/xkosze00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Köszegy Lukáš, Bc.**
Obor: Management a informační technologie
Téma: **Automatizované testování webových aplikací**
Automated of Web Application Testing
Kategorie: Analýza a testování softwaru

Pokyny:

1. Seznamte se s možnostmi automatizovaného testování v rámci agilního vývojového procesu.
2. Specifikujte požadavky na testovací plán pro rozsáhlejší online aplikaci sloužící pro vytváření a provozování online obchodů.
3. Diskutujte jednotlivé úrovně testovacího plánu a v souladu s testovacím plánem navrhněte vhodné řešení pro funkční testování.
4. Zvolte vhodné vývojové prostředí a implementujte prototyp navrženého systému pro automatické funkční testování ve vybrané aplikaci.
5. Použitelnost vytvořeného prototypu nástroje otestujte v reálném prostředí a demonstrujte na vhodně zvoleném vzorku dat vybraném po dohodě s vedoucí.
6. Zhodnoťte kvalitu výsledného řešení především s ohledem na stabilitu a spolehlivost testů. Diskutujte další možná rozšíření.

Literatura:

- AMMANN, Paul a Jeff OFFUTT. *Introduction to software testing*. New York: Cambridge University Press, 2008. ISBN 978-0-521-88038-1.
- CRISPIN, Lisa. a Janet GREGORY. *Agile testing: a practical guide for testers and agile teams*. Upper Saddle River, NJ: Addison-Wesley, c2009. Addison-Wesley signature series. ISBN 978-0-321-53446-0.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů zadání 1 až 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kreslíková Jitka, doc. RNDr., CSc.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Diplomová práca sa zameriava na základný popis teórie a problematiky testovania, agilného vývoja a internetových obchodov. Cieľom práce je špecifikovať požiadavky na testovací scenár internetových obchodov a následne navrhnúť vhodné aplikačné riešenie pre jednoduchú integráciu testovacieho procesu ako súčasť vývoja internetových obchodov v rámci agilného vývoja.

Abstract

The Master thesis focuses on basic description of the theoretical and practical part of testing, agile development and online shopping. Aim of the paper is to specify requirements for testing scenario of online shopping and subsequently suggest suitable solution for easy integration testing process as a part of the development of online shopping within agile development.

Kľúčové slová

testovanie, e-shopy, web stránky, react, rest, server, webové rozhranie, klient, regresné testy, výkonnostné testy, modularita

Keywords

testing, e-shop, web pages, react, rest, backend, frontend, client, regress tests, performance tests, modularity

Citácia

KÖSZEGY, Lukáš. *Automatizované testování webových aplikací.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. RNDr. Jitka Kreslíková, CSc.

Automatizované testování webových aplikací

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pani doc. RNDr. Jitky Kreslíkové CSc.. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Lukáš Köszegy
17. mája 2018

Poďakovanie

Týmto by som chcel poďakovať vedúcemu práce za odborné konzultácie, venovaný čas a poskytnutú literatúru.

Obsah

Kapitola 1

Úvod

S rozšírením internetu sa objavilo mnoho nových technológií a odvetví, ktoré využívajú internet. Jednou z dôležitých technológií pre výmenu informácií bol **W**orld **W**ide **W**eb (WWW). WWW bol vytvorený Timom Berners-Lee v roku 1989 za účelom zdieľania vedeckých informácií[5]. Tieto informácie sú prezentované formou dokumentov (webových stránok), ktoré sú uložené na webovom servery. Pre prezeranie týchto dokumentov na klientskej strane slúžia aplikácie, ktoré sa nazývajú webové prehliadače.

S rozšírením WWW a počítačov aj do iných odvetví a do domácností bežných ľudí vzniklo nové odvetvie trhu predaja produktov a služieb. Toto odvetvie trhu sa nazýva elektronické obchodovanie. Ide o spôsob realizácie obchodnej transakcie s využitím elektronických komunikačných prostriedkov. V práci sa zameriavam na podmnožinu elektronického obchodovania, ktoré sa nazýva internetový obchod (e-shop). Pri internetovom obchode ide o vytvorenie obchodnej transakcie s využitím webových stránok, ktoré sa nazývajú e-shopy a špecializujú sa na predaj produktov a služieb.

Cieľom tejto práce je testovanie vyššie zmienených e-shopov. Testovanie je dôležitou súčasťou vývoja. V práci sa zameriavam na testovanie e-shopov vyvíjaných s využitím agilných metodík. Primárnym spôsobom testovania bude takzvané black box testovanie, ktoré sa zameria na testovanie funkčnosti e-shopov pomocou regresného testovania.

Výsledná aplikácia bude postavená na 3-vrstvej architektúre pozostávajúcej z častí **Server**, **Klient** a **Webové rozhranie**. Hlavným cieľom aplikácie je ľahká tvorba testovacích scenárov a ich následná aplikácia v testovacom procese. Jednotlivé časti aplikácie sú navrhnuté s ohľadom na vysokú modularitu a to hlavne za účelom ľahkého rozširovania aplikácie a jej samotného testovania. Ovládacie rozhranie aplikácie bude postavené na webových technológiach, aby bolo možné aplikáciu využívať na rôznych platformách.

Diplomová práca priamo pokračuje v semestrálnom projekte, kde v 2 kapitole je základný popis elektronických obchodov spoločne so stručným opisom využívaných technológií pri ich vývoji. V nasledujúcej 3 kapitole sa bližšie venujem agilným metodikám a teórií z oblasti testovania aplikácií. Následne v kapitole 4 Návrh je definovaný testovací scenár spoločne s vysoko úrovňovou architektúrou celej aplikácie, ktorá bola oproti semestrálnemu projektu zjednodušená. Za návrhom nasledujú už rozšírenia napísane ako súčasť diplomovej práce. Kapitola 5 Implementácia podrobnejšie popisuje jednotlivé častí aplikácie. V kapitole 6 Testovanie sú popísané postupy a výsledky pre overenie funkčnosti a použiteľnosti aplikácie. Celkový súhrn je potom uvedený v záverečnej kapitole 7 Záver.

Kapitola 2

Internetové obchody

Internetové obchody, tiež elektronické obchody (skrátene: e-shopy) sú podmnožina elektronického obchodovania. Elektronické obchodovanie je obchodná transakcia, ktorá je realizovaná pomocou elektronických komunikačných prostriedkov. V rámci elektronického obchodovania môžeme rozlišovať tieto základe druhy obchodných vzťahov:

Skratka	Celý názov	Význam
B-2-B	obchodník obchodníkovi	ponuka jednej firmy inej firme
B-2-C	obchodník zákazníkovi	obchodná ponuka pre zákazníka
B-2-G	obchodník štátnej správe	ponuka služieb a produktov
C-2-B	zákazník obchodníkovi	obchodnú transakciu inicializujú kupujúci
C-2-C	zákazník zákazníkovi	predaj medzi koncovými užívateľmi
C-2-G	zákazník štátnej správe	napríklad podanie daňového priznania
G-2-G	štátna správa štátnej správe	spolupráca štátnych orgánov

Tabuľka 2.1: Základné typy obchodných vzťahov (inšpirované [17])

E-shopy sú podmnožina, kde ako komunikačný nástroj medzi predajcom a nakupujúcim sú využívané webové stránky. Rovnako, ako elektronické obchodovanie môžu byť e-shopy využité pre všetky druhy obchodných vzťahov. Najväčšie zastúpenie má však typ B-2-C, kde na danom modely existujú celé firmy.

Výhoda e-shopov oproti klasickým kamenným obchodom spočítava primárne v znížení nákladov a rozšírenie oblasti pôsobnosti. E-shopy oproti kamenným obchodom s využitím kuriérov môžu ponúkať svoje produkty v celej republike ale tiež je možné exportovať produkty do celého sveta. Zároveň sú nižšie náklady na predajcov aj na samotnú predajňu. Preto e-shopy predstavujú dobrú možnosť rozšírenia obchodnej pôsobnosti na väčšie množstvo potenciálnych zákazníkov.

Nakoľko sú webové stránky pre e-shopy primárnou formou kontaktu so zákazníkmi, je kladený väčší dôraz na kvalitu a vizuálnu prezentáciu. Z toho dôvodu môžu byť e-shopy navrhnuté ako komplexné webové aplikácie s veľkou radou funkcií a rôznych technológií, aby pôsobili pre zákazníka atraktívne a predajca získal určitú konkurenčnú výhodu na trhu. To ale vedie k celkovej zložitosti celého e-shopu. Zároveň elektronické obchodovanie podlieha zákonom štátu v ktorom primárne pôsobí.

Všetky vyššie spomenuté aspekty pôsobia na celkový vývoj a samotných vývojárov webových aplikácií, ktorí majú zodpovednosť za výslednú kvalitu webovej stránky. Aby

bolo možné dosiahnuť požadovanú kvalitu a eliminovať nežiadúce chovanie je nutné do vývojového procesu zaradiť určitú formu testovania.

E-shopy sú primárne postavené na webových technológiách. Nevýhodou pri webových technológiách je ich veľký počet a ich možné kombinácie. Zároveň vývoj niektorých webových technológií je značne nestabilný a stáva sa, že vzniká veľké množstvo technológií. Niektoré z nich po čase zaniknú. V súčasnosti najpoužívanejšie technológie pri vývoji webových stránok sú HTTP/S, Hypertext Markup Language (HTML), Cascading Style SHeets (CSS), JavaScript, REST, PHP a SQL databázy.

2.1 HTTP

Hyper**T**ext **T**ransfer **P**rotocol (HTTP) je komunikačný protokol na aplikačnej vrstve ISO/OSI modelu. Primárne bol navrhnutý pre výmenu hypertextových dokumentov vo formáte HTML. Tieto dokumenty sú identifikované pomocou Uniform Resource Locator (URL), ktorá definuje umiestnenie dokumentu v sieti.

Samotný HTTP protokol je definovaný v RFC 2616 [11]. Okrem prenášania hypertextových protokolov je dnes možné pomocou rozšírenia MIME využívať HTTP protokol aj pre prenos súborov.

HTTP bol navrhnutý ako bezstavový protokol vo forme dotaz-odpoveď. Najčastejšie server počúva na porte 80, kde očakáva HTTP dotaz na konkrétny dokument identifikovaný pomocou URL. HTTP je možné používať v kombinácií s SSL/TLS pre zabezpečenie komunikácie. V takom to prípade hovoríme o protokole HTTPS, ktorý je väčšinou dostupný na porte 443.

2.1.1 URL

Uniform Resource Locator (URL) je podmnožina Uniform Resource Identifier (URI), ktorá okrem URL obsahuje aj Uniform Resource Name (URN) [7]. URL špecifikuje umiestnenie dokumentu v sieti a zároveň spôsob, ako daný dokument získať. Je špecifikovaná v RFC 1738 [7] a jej základná schéma vychádza z URI:

```
1 scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment]
```

Výpis 2.1: URL vzor (prevzaté [6])

- **scheme** – Definuje konkrétnu schému URI. V prípade protokolu HTTP to môže byť http alebo https
- **:** – Oddeľovač
- `//` – Sú vyžadované pre niektoré schémy. V prípade protokolu HTTP sú povinné.
- **Autorizačná časť** - Je voliteľná súčasť a môže slúžiť pre autentifikáciu užívateľa pomocou mena a hesla pri využití základnej HTTP autorizácie. Užívateľské meno je od hesla oddelené pomocou znaku **:**. Prihlasovacie údaje sú potom oddelené od hosta pomocou znaku **@**.
- **host** – Identifikátor webového servera. Môže obsahovať DNS záznam alebo IP adresu.
- **port** – Voliteľná súčasť, ktorá je od hosta oddelená pomocou znaku **:**. Využíva sa v prípade, že server počúva na neštandardnom porte.

- **path** – Každá cesta začína lomítkom (/) a špecifikuje umiestnenie dokumentu na webovom servery.
- **query** – Je voliteľná súčasť oddelená od cesty znakom **?**. Ide o niehierarchické dáta vo formáte: kľúč=hodnota. Jedna URL môže obsahovať viac dát, ktoré sú medzi sebou oddelené znakmi **&** alebo **;**.
- **fragment** – Je oddelený od ostatných súčasti pomocou znaku # a slúži najčastejšie k špecifikovaniu konkrétnej sekcie v dokumente. V prípade HTML fragment reprezentuje atribút id v elementoch, čo umožňuje napríklad automatické skrolovanie na daný element v dokumente.

2.1.2 Správy

HTTP správy pozostávajú z nasledujúcich 2 hlavných častí:

- **Hlavička** – obsahuje informácie vyžadované protokolom HTTP ako metóda, host, dĺžka tela správy, ale môže obsahovať aj užívateľom definované atribúty vo formáte kľuč: hodnota. Táto časť správy nie je zobrazovaná koncovému užívateľovi a využíva sa pre prenos informácií určených webovému serveru, alebo webovému prehliadaču.
- **Telo správy** – informácie, ktoré sa primárne interpretujú koncovému klientovi. V prípade e-shopov môže ísť o samotné webové stránky.

Dôležité sú pri HTTP metódy dotazov, tie najdôležitejšie pre webové stránky sú nasledujúce [11]:

- **GET** – Metóda pre požiadanie na zobrazenie hypertextového dokumentu, ktorá umožňuje zaslanie dát na server.
- **HEAD** – podobná metóda ako GET, ale môže prenášať na server len metadata.
- **POST** – Metóda primárne určená pre odosielanie užívateľských dát na server. Na rozdiel od metódy GET je určená pre prenos veľkého objemu dát, ktoré nie je možné preniesť metódou GET.
- **PUT** – Slúži primárne pre nahranie súboru na server.
- **DELETE** – Zmaže uvedený objekt zo servera.

2.2 HTML

Hyper**T**ext **M**arkup **L**anguage (HTML) je značkovací jazyk používaný pre tvorbu webových stránok [19]. Vychádza zo značkovacieho jazyka **S**tandard **G**eneralized **M**arkup **L**anguage (SGML). Jazyk je charakterizovaný množinou elementov (tagov) a ich vlastností (atribútov), ktoré sú definované pre danú verziu jazyka HTML. HTML jazyk je štandardizovaný a pod správou World Wide Web Consortium (W3C). Aktuálna verzia je HTML5.2.

2.2.1 Štruktúra dokumentu

Dokumenty v jazyku HTML majú predpísanú štruktúru, ktorá by mala byť v rámci štandardu HTML dodržiavaná, ale webové prehliadače akceptujú aj neštandardnú štruktúru.

Doporučená štruktúra dokumentu [20]:

- **Typ dokumentu** – (<!DOCTYPE html>) informuje webový prehliadač, že pracuje s HTML dokumentom. Môže špecifikovať aj konkrétnu použitú verziu jazyka HTML.
- **Koreňový element** – (<html></html>) reprezentuje html dokument.
- **Hlavička dokumentu** – (<head></head>) obsahuje metadata, ktoré sa vzťahujú na celý dokument. Môže obsahovať napríklad definovanie kódovania dokumentu, názov, autora, atď.
- **Telo dokumentu** – (<body></body>) zahrňuje obsah dokumentu, ktorý je zobrazovaný užívateľovi.

2.3 CSS

Cascading **S**tyle **S**heets (CSS) je jazyk pre popis formátovania HTML dokumentu [19]. Jazyk je spravovaný tiež World Wide Web Consorciom a bol navrhnutý pre oddelenie popisu vizuálnej prezentácie dokumentu od jeho štruktúry. CSS jazyk je pomocou selektorov prepojený s HTML dokumentov. Ako selektory sa pre CSS používajú:

- **Elementy** – Každý typ elementu môže byť použitý ako selektor (body, h1, atď).
- **Atribúty**:
  - **ID** – unikátny identifikátor elementu
  - **Class** – identifikátor, ktorý môže byť zdieľaný medzi viacerými elementami.
- **Závislosti** – Je možné použiť relatívne umiestnenie v rámci štruktúry dokumentu.
- **Pseudo-triedy** – Sú špeciálne identifikátory definované pre určité typy chovania na webovej stránke (prechod myšou nad elementom).

2.4 JavaScript

JavaScript je vysoko úrovňový, dynamický, slabo typovaný, prototypový, multiparadigmatový a interpretovaný programovací jazyk. Javascript je štandardizovaný ako ECMAScript, ktorý spravuje štandardizačná organizácia **E**uropean **C**omputer **M**anufacturers **A**ssociation (ECMA). Aktuálna verzia je ECMAScript6 (ES6) [10]. Javascript je v dnešnej dobe mnoho účelový jazyk, ktorý podporuje veľké množstvo platforiem. Jeho využitie v rámci vývoja webových stránok je možné ako súčasť samotných webových stránok, kde sa primárne používa pre pridanie dynamických vlastností pre obsah. Okrem toho je dnes možné použiť Javascript aj na strane servera, kde sa používa primárne s behovým prostredím Node.js a môže tak slúžiť ako náhrada za jazyky PHP, Python a Java.

2.5 REST

Representation **S**tate **T**ransfer (REST alebo RESTful) umožňuje ľahko implementovať CRUD (Create, Read, Update a Delete) pre informácie na servery pomocou protokolu HTTP [9]. REST slúži ako jednotné a jednoduché rozhrania služieb, ktoré na servery spravujú zdroje. Jednotlivé zdroje sú identifikované pomocou URI a pre prácu s nimi sa používajú najčastejšie metódy GET, POST, PUT, DELETE a OPTIONS.

Samotný REST je oproti XML-RPC a SOAP dátovo orientovaný. Pri výmene dát sa väčšinou pracuje s určitým reprezentačným formátom dát ako je JavaScript Object Notation (JSON), Extensible Markup Language (XML), HTML, alebo inými definovanými formátmi. Nevýhodou tohto riešenia oproti SOAP je chýbajúca podpora pre middleware a REST je plne orientovaný na server.

Medzi hlavné výhody RESTful služieb patrí:

- **Distribuovaná architektúra** – Funkcionalita môže byť rozdelená medzi viacerými servermi.
- **Bezstavovosť** – Požiadavka musí obsahovať všetky informácie potrebné k jeho vykonaniu.
- **Jednotnosť** – Využitie CRUD pre prácu so zdrojmi a jednotný popis rozhrania.
- **Reprezentácia dát** – Dáta môžu byť reprezentované v rôznych formátoch.
- **Code-On-Demand** – Rozšírenie funkcionality klienta s využitím kódu zaslaného zo servera.
- **Rozšíriteľnosť** – Jednoduché pridávanie novej funkcionality aj za behu.
- **Výkon** – Jednoduchosť a kombinácia minimalistických formátov dát (oproti SOAP).
- **Portabilita** – Ľahký presun kódu spoločne s dátami.
- **Klient-Server** – Komunikácia na základe HTTP protokolu typu dotaz-odpoveď.
- **Cache** – Využitie cachevania odpovedí pre minimalizáciu nevýhod architektúry klient-server.

Kapitola 3

Vývoj aplikácií

Vývoj aplikácií je súhrny názov pre všetky procesy tvorby programov pozostávajúcich z programovania, dokumentácie, testovania a opravy chýb. Zároveň, ale vývoj aplikácií môže tiež pozostávať z výskumu, prototypovania, modifikovania, analýzy alebo iných aktivít, ktorých výsledkom bude aplikácia. Ako súčasť vývoja aplikácií sa pre definovanie procesu vývoja v dnešnej dobe vo veľkom presadili takzvané agilné metodiky, ktoré sú zamerané na rýchly vývoj aplikácií (RAD).

3.1 Agilný vývoj

Agilný vývoj je založený na takzvaných agilných metodikách, ktoré popisujú vývoj aplikácií v iteratívnych alebo inkrementálnych cykloch [8]. Hlavnými výhodami agilného vývoja je rýchly vývoj a schopnosť reagovať na zmenu požiadaviek počas vývojového cyklu. Agilné metodiky sa opierajú o nasledujúce 4 hlavné hodnoty [8]:

- **Spolupráca** – Komunikácia, motivácia a samostatnosť sú dôležitejšie ako znalosti a procesy.
- **Fungujúca aplikácia** – Uprednostňovanie fungujúcej aplikácie pred dokumentáciou.
- **Zákazník** – Zapájať zákazníka do celého vývojového cyklu a pravidelne s ním komunikovať.
- **Zodpovednosť za zmeny** – Uprednostniť zmenu pred plánom.

Existuje veľké množstvo agilných metodík, ktoré sú dnes v praxi používané. Nevýhodou alebo výhodou týchto agilných metodik je, že nie sú striktne definované a dodržiavané. Rôzne vývojové tímy si prispôsobujú metodiky pre vlastné potreby [8]. Medzi najznámejšie metodiky patrí Extrémne programovanie (XP), Scrum, Crystal metodika, Vývoj riadený testami (TDD), Lean development a Vývoj riadený vlastnosťami (FDD).

3.1.1 Extrémne programovanie (XP)

XP stavia na krátkych vývojových cykloch s využitím párového programovania [14]. Ďalšie faktory na ktoré XP spolieha sú:

- **Unit testy (3.2.3)** – Pravidelné a automatizované testy funkčných časti kódu.
- **Refactoring** – Pravidelná kontrola kódu inými programátormi.

- **Jednoduchosť** – Jednoduchý a jasný kód.

3.1.2 Scrum

Iteračné cykly v scrume sa volajú sprinty a trvajú 1-4 týždne. Výsledkom sprintu by mala byť demo aplikácia s požadovanými úpravami, ktorá je prezentovaná zákazníkovi. Súčasťou scrumu sú aj denne standups, na ktorých programátori prezentujú čo za minulý deň stihli spraviť, aké riešili problémy a čo budú robiť aktuálny pracovný deň. Scrum naviac definuje aj 3 úlohy v rámci vývoja:

- **Vlastník produktu** – Je prostredník medzi zákazníkom a vývojovým tímom.
- **Scrum majster** – Zabezpečuje správne fungovanie vývojového tímu.
- **Člen scrum tímu** – Člen vývojového tímu.

3.1.3 Lean development

Ide o súhrn pravidiel, princípov a nástrojov, ktorých cieľom je zefektívniť a urýchliť vývoj. Do Lean development patrí napríklad: eliminácia zbytočného, posilnenie zodpovednosti tímu za výsledný produkt a mnoho ďalších aspektov.

3.1.4 Crystal metodika

Princíp spočíva vo výbere/definovaní vhodnej metodiky pre potreby daného projektu. Nejde teda o jednu konkrétnu metodiku, ale o vytvorenie individuálnej a účelovej metodiky pre konkrétny projekt za účelom získania čo najlepších výsledkov na základe prostredia daného projektu.

3.1.5 Vývoj riadený vlastnosťami (FDD)

FDD narozdiel od XP a Scrumu sa viac zameriava na plánovanie a riadenia. V prípade FDD sú úlohy delegované programátorom. FDD preto vývoj rozdeľuje do 5 etáp, kde etapy 1 až 3 sú sekvenčné a nasledujúce (4 a 5) sú iteratívne:

1. **Tvorba celkového modelu** - Tvorba vysoko úrovňového modelu daného systému s využitím CASE nástrojov.
2. **Zoznam vlastností** - Na základe modelu sa vytvorí zoznam vlastností systému, ktoré by nemali byť rozsiahle a majú prinášať zákazníkovi hodnotu.
3. **Plánovanie** - Na základe zoznamu vlastností je vytvorený vývojový plán so zadefinovanými vlastníkmi, ktorý zodpovedajú za implementáciu vlastnosti.
4. **Návrh vlastností** - Počas jednej iterácie vyberie hlavný vývojár malú skupinku vlastností, ktoré by mali ich vlastní vývojári v priebehu 2 týždňov implementovať.
5. **Zostavenie vlastností** - Finálny krok pri ktorom kód vlastností predchádza inšpekciou (Refactoring) a testovaním s využitím unit testov. Ak nie sú žiadne problémy, môže byť vlastnosť zaradená do výsledného programu.

3.1.6 Vývoj riadený testami (TDD)

Rovnako ako FDD definuje TDD 5 etáp, ale počas jednej iterácie, ktoré sú pravidelne opakované s určitými prírastkami. TDD ako z názvu vyplýva sa zameriava na testy a tomu sú podriadené aj jednotlivé etapy:

1. **Tvorba testov** - Tvorba testov, ktoré overujú požiadavky na funkcionalitu danej časti kódu.
2. **Prvé spustenie testov pre kontrolu** - V prípade prvého spustenia sa testuje funkčnosť samotných testov a to tak, že by každý test mal skončiť neúspešne.
3. **Písanie kódu** - Tvorba kódu primárne zameraná na rýchly vývoj a aby kód prešiel testami.
4. **Automatické testovanie** - Pravidelné testovanie kódu pomocou automatizačných nástrojov, pre overenie či nebola počas zmeny kódu zanesená chyba do kódu.
5. **Refaktorizácia** - Optimalizácia a prečistenie kódu s ohľadom na efektívnosť. S využitím automatických testov je možné v tomto kroku testovať či nedošlo k zmene požiadaviek na funkčnosť výsledného kódu.

3.2 Testovanie

Testovanie je proces zisťovania kvality produktu alebo služby za účelom získania určitých informácií o danom produkte alebo službe. Proces testovania sa v určitej forme preťahuje do celého procesu vývoja. Obzvlášť v prípade agilných metodík testovanie predstavuje samostatnú etapu vývoja programov.

Hlavným cieľom testovania je odhaliť chyby v programoch, ktoré vedú na nepovolené alebo neočakávané chovanie. K tomuto účelu väčšina výsledkov testov má určité ohodnotenie. Na testovanie programov sa vo väčšine využívajú dve stupňové ohodnotenia:

- **Prešiel (Bezchyby)** – Ohodnotenie znamená, že počas testu nebola odhalená žiadna chyba a výstup testovaného produktu zodpovedá akceptovateľnému výsledku.
- **Neprešiel (Chyba)** – Počas testu produktu boli detekované chyby, alebo výsledný výstup testovaného produktu neodpovedá akceptovateľnému výsledku.

Proces testovania je značne náročný a dochádza v ňom tiež k chybám. Medzi najznámejšie chyby patria:

- **false positive** – Test akceptuje nesprávne výsledky testovaného produktu a nehlási chybu.
- **false negative** – Takzvaný falošný poplach. Je opak false positive, keď pre korektný výstup testovaného produktu test neprešiel a bola detekovaná chyba.

Tieto chyby vychádzajú priamo z princípu testov, kde nie je možné v rámci testov simulovať všetky možné vstupy a následne skontrolovať vygenerované výstupy. Zároveň je značne zložité otestovať všetky súčasti aplikácií. K týmto problémom s testovaním zároveň prispievajú aj následujúce faktory:

- Nejednoznačná alebo chýbajúca špecifikácia.
- Zlé pochopenie alebo podcenenie testov.
- Neaktuálnosť testov a meniace sa okolnosti.
- Testovacie dáta neodpovedajú reálnym dátam.
- Nedostatočná alebo zlá komunikácia v tíme.
- Testy by mali byť špecifické pre každý produkt.
- Rôzne chyby sa môžu prejavovať rovnako.
- Jedná chyba sa môže prejavovať rôzne.

3.2.1 Dimenzie testov

Testy rozdeľujeme do určitých kategórií na základe aspektov vyvíjaného produktu alebo služby, ktoré sú počas procesu testované. Tieto kategórie sú vhodné pri plánovaní a vytváraní testov.

- **Bezchybnosť** – Testovanie chovania na základe funkčnej špecifikácie [14].
- **Použiteľnosť** – Tieto testy sa používajú primárne pre testovanie užívateľského rozhrania a jeho použiteľnosť samotnými užívateľmi [14].
- **Spoľahlivosť** – Spoľahlivosť sa väčšinou testuje penetračnými testami. V týchto testoch sa overuje správna funkcionalita počas všetkých okolností hlavne v prípade preťaženia, výpadkov alebo pri vzniku poruchy [14].
- **Efektivita** – Overujú sa požiadavky na systémové zdroje hlavne pre viac pracujúcich užívateľov [14].
- **Udržovateľnosť** – Testovanie inštalácie, konfigurácie a iných vlastnosti alebo činnosti, ktoré súvisia s údržbou a aktualizáciou produktu alebo služby [14].

Tieto vyššie uvedené rozdelenia testov sú tiež označované ako FURPS [1] a vychádzajú z medzinárodného štandardu ISO/IEC 9126, ktoré popisuje kvalitu produktov ako súčasť Softwarového inžinierstva.

3.2.2 Kategórie testov

Kategórie testov sú rovnako ako dimenzie testovania používané pri plánovaní, komunikácií a návrhu. Tieto kategórie rozdeľujú testy na základe spôsobu testovania produktov a služieb.

Statické a dynamické testy

Testy sa rozdeľujú na statické a dynamické na základe toho či je potrebné, aby testovací produkt alebo služba počas testu boli spustené.

Statické testy nevyžadujú počas testovania aby testovaný subjekt bol spustený. Tieto testy sa často používajú už počas programovacieho procesu, kde sú integrované do IDE a môžu robiť napríklad typovú kontrolu a využitie zadefinovaných premenných [3].

Dynamické testy vyžadujú spustenie programu a umožňujú väčšie množstvo testov. Sú postavené väčšinou na poskytovaní rôznych druhov vstupu a testovaní výstupov z programu [3].

Čierna skrinka

Ide o typy testov pri ktorých nieje známa implementácia testovaného produktu alebo služby [3]. Tieto typy testov sa preto zameriavajú na vstupy a výstupy, na vonkajšie chovanie testovaného produktu alebo služby. Pri tomto type testu je produkt alebo služba testovaná z pohľadu užívateľa.

Biela skrinka

Hlavný rozdiel v porovnaní s čiernou skrinkou je v tom, že tester pozná vnútornú implementáciu alebo má prístup k zdrojovým kódom testovaného produktu alebo služby [3]. Tento typ testovania umožňuje lepšie zameranie testov na kritické časti a minimalizuje veľkosť testovacích dát podľa potreby. Nevýhodou oproti čiernej skrinke je takzvaná strata užívateľského pohľadu, ktorý je veľmi dôležitý.

Siva skrinka

Sivá skrinka je medzistupeň medzi čiernou a bielou skrinkou. Tester v tomto prípade má určité informácie o implementácií (použité algoritmy, vzorce, atď) ale nemá napríklad k dispozícií zdrojové kódy a teda nejde o bielu skrinku. Oproti čiernej skrinke sú ale známe určite informácie, ktoré umožňujú lepšie špecifikovať vstupné testovacie dáta.

A/B testy

Základom je porovnávanie dvoch výstupov na zmenu medzi 2 testami. Cieľom týchto testov je jednoduché odhalenie neočakávaných zmien.

Testovanie krajných hodnôt

Ako vyplýva z názvu, vstupné hodnoty pre testy sa používajú určité okrajové hodnoty a ich okolie, ktoré vyplývajú z implementácie alebo špecifikácie. Napríklad, ak vieme, že funkcia používa 32 bitové celé číslo uložené v premennej na vstupe, otestujú sa hraničné hodnoty, ktoré môže daná premenná nadobudnúť a zároveň sa tiež otestujú hodnoty presahujúce tento limit, aby sa overilo či je vstup ošetrený a nedochádza k pretečeniu.

Automatické a manuálne testovanie

Automatické testovanie sa od manuálne rozlišuje podľa toho, či testy vykonáva program alebo človek.

Automatické testovanie je veľmi dôležitou súčasťou pre zvýšenie efektívnosti. Všetky testy, ktoré je možné testovať automaticky a nevyžadujú zásah človeka, by mali byť automatizované [8]. Automatické testy sú veľmi doporučované hlavne ako súčasť agilných metodík a to hlavne v metodike TDD.

Manuálne testovanie by malo vždy predchádzať automatickému testovaniu a to hlavne z dôvodu overenia samotných testov. Využitie manuálneho testovania by malo byť len v prípadoch, keď nie je možné aplikačne zabezpečiť testovanie a v prípade tvorby nových testov [8].

3.2.3 Úrovne testov

Testy môžme taktiež rozdeliť podľa toho, ktorú časť a veľkosť produktu alebo služby testujú.

Jednotkové testy

Tiež nazývané ako Unit testy sú testy písane vývojármi pre overenie určitých časti kódu ako sú napríklad funkcie alebo triedy [14]. Jednotkové testy spadajú do kategórie testov bielej skrinky, nakoľko je nám známa implementácia a tester (vývojár) má prístup k zdrojovým kódom.

Jednotkové testy ale majú určité obmedzenia vychádzajúce z ich definície:

- **Neaktuálnosť** – Nakoľko sú testy závislé na implementácií, je nutné pri zmene implementácie aktualizovať aj testy.
- **Obmedzenosť** – Nakoľko sú testy určené pre testovanie určitých časti kódu, zlyhávajú pri integrácií testovaných časti kódu do iných aplikácií alebo prostredí.

Integračné testy

Integračné testy vhodne doplňujú jednotkové testy pri integrácií komponentu do väčších celkov. Cieľom testov je primárne testovanie rozhraní medzi jednotlivými komponentami, ktoré medzi sebou komunikujú [14]. Integračné testy sa zaoberajú primárne funkcionalitou a spadajú do kategórie čiernej skrinky.

Ako súčasť integračných testov sa využívajú nasledujúce prístupy:

- **Zdola-nahor** – Je počas vývoja najčastejšie používaný prístup, nakoľko priamo nadväzuje na jednotkové testy. Po vykonaní jednotkových testov sa testujú postupne väčšie časti produktu alebo služby, kde posledný krok je testovanie produktu alebo služby ako celku [3].
- **Zhora-nadol** – Tento prístup je vhodný pre hľadanie chýb. Ide o opačný postup ako pri prístupe Zdola-nahor, kde sa začína testovaním celého produktu alebo služby ako celku a následne sa postupne testujú menšie a menšie časti [3].

Regresné testovanie

Je založené na previazanosti kódu a vplyvoch zmien. Regresným testovaním ide hlavne o overenia zachovania funkčnosti po aplikovaní zmien v zdrojovom kóde (napríklad oprava chyby) [14]. To je dôležité, nakoľko zmena jednej časti kódu môže narušiť funkčnosť inej časti kódu.

Systémové testovanie

Počas systémového testovania sa testuje produkt alebo služba ako celok. Cieľom je otestovanie produktu alebo služby v podobnom prostredí ako cieľové prostredie a overiť tak požadovanú kvalitu a nároky na zdroje [14].

Akceptačné testovanie

Tento typ testov je väčšinou viazaný na špecifikáciu a overuje či produkt alebo služba spĺňa požiadavky klienta [8]. Tieto požiadavky sú vo väčšine prípadov definované v zmluve medzi obchodníkom a zákazníkom a na základe ich výsledkov sa rozhoduje či zákazník prevezme produkt alebo službu. Preto sú tieto testy veľmi dôležité a v ich rámci sa testuje dôležitá funkcionalita, ktorá v prípade chyby by mala za následok poškodenie produktu alebo zákazníka. Akceptačné testy môžu tiež vychádzať zo systémových testov a rozširovať ich o zmluvné požiadavky na funkčnosť, ale dôležité pri akceptačných testoch je testovacie prostredie. To by malo byť v prípade akceptačných testov výhradne produkčné.

Kapitola 4

Návrh

Pri návrhu aplikácie sa brali do úvahy informácie z kapitoly 2. Internetové obchody a 3. Vývoj aplikácií. Na základe týchto informácií sa určili hlavné ciele, ktoré by mali byť dosiahnuté ako súčasť výslednej aplikácie.
Ciele na výslednú aplikáciu:

- **Jednoduchosť**
- **Rozšíriteľnosť**
- **Automatizovanie**
- **Použiteľnosť**

V testoch som bral hlavne ohľad na proces testovania e-shopov. Pri návrhu špecifikácií pre testovací plán som sa zameral na samotný vývojový cyklus, technológie pre vývoj e-shopov, problematiku a zložitosť samotného testovania.
Špecifikácia pre testovací plán:

- **Zameranie na základnú funkčnosť e-shopov**
  - **Prihlásenie**
  - **Vyhľadávanie**
  - **Katalóg produktov**
  - **Objednávky**
- **Nezávislosť na použitých technológiách**
- **Užívateľský pohľad**
- **Spoľahlivosť**
- **Pružnosť** a **Modifikovateľnosť**

4.1 Testovací scenár

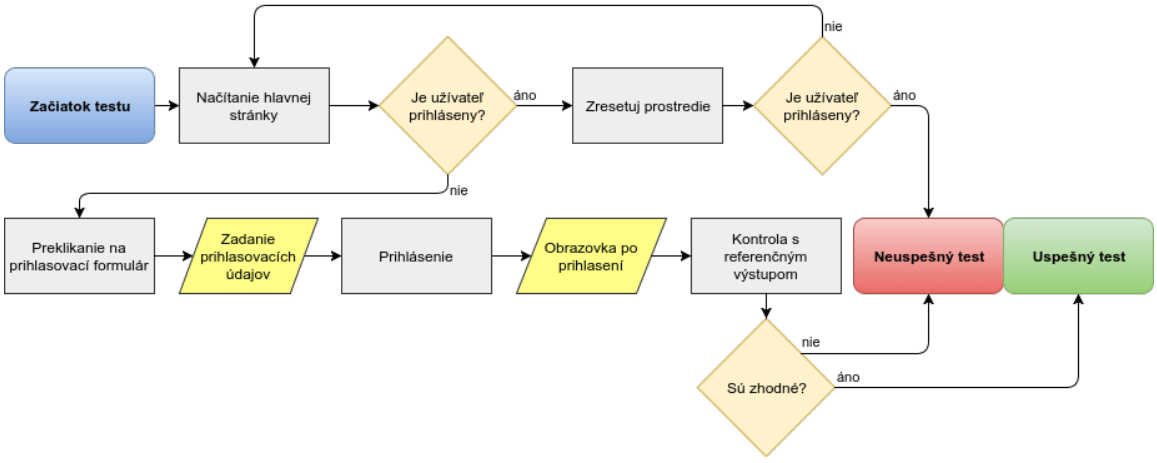
Pre aplikáciu sú dôležité testovacie scenáre. V tomto prípade testovacie scenáre sú značne závislé na type testov. Tu je ale značný problém na strane testovaného produktu alebo služby. Ako bolo spomenuté v kapitole 2. Internetové obchody, väčšina e-shop je postavená na rôznych technológiách. To značne komplikuje návrh testov, ktoré sú viazané na implementáciu. Kvôli tomuto budem v testovacom scenári používať testy z kategórie Čierna skrinka (3.2.2). Ďalej sa zameriam na dimenziu kvality Funkčnosť (3.2.1) ako súčasť testovacích scenárov, v ktorej je možné testy ľahšie kvantifikovať. Pri kvantifikácií testov využijem 2 úrovňové hodnotenie výsledku testov vo forme Prešiel/Neprešiel.

Testovací scenár by sa mal skladať zo 4 testov, ktoré sú kvôli veľkým rozdielom v užívateľskom rozhraní a ovládaní e-shop napísane na vyššej abstrakčnej úrovni.

- **Prihlásenia užívateľa**
- **Tvorba objednávky 1 produktu**
- **Tvorba objednávky viacerých produktov**
- **Zrušenie objednávky**

4.1.1 Prihlásenie užívateľa

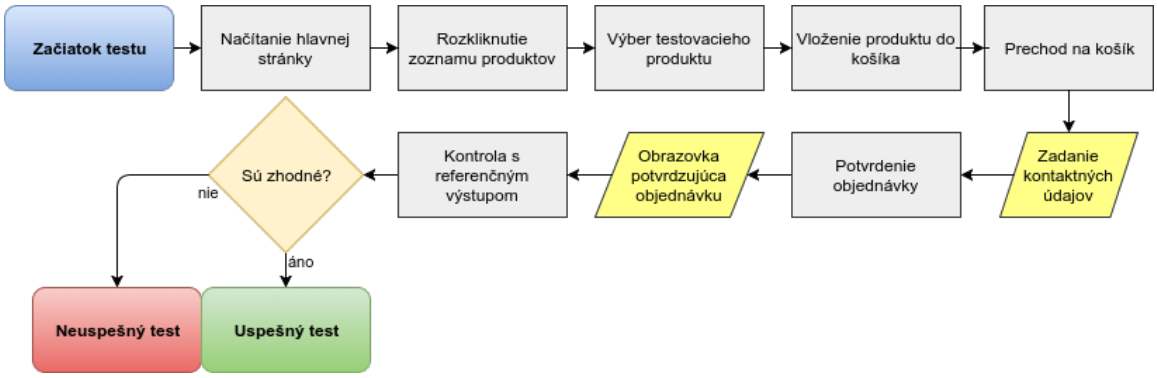
Kategória: Čierna skrinka
Dimenzia: Funkčnosť
Oblasť: Zoznam produktov
Metrika: Porovnávanie obrázkov okna prehliadača na zhodu s referenčnými obrazmi



Obr. 4.1: Test prihlásenia užívateľa (vlastné)

4.1.2 Tvorba objednávky jedného produktu

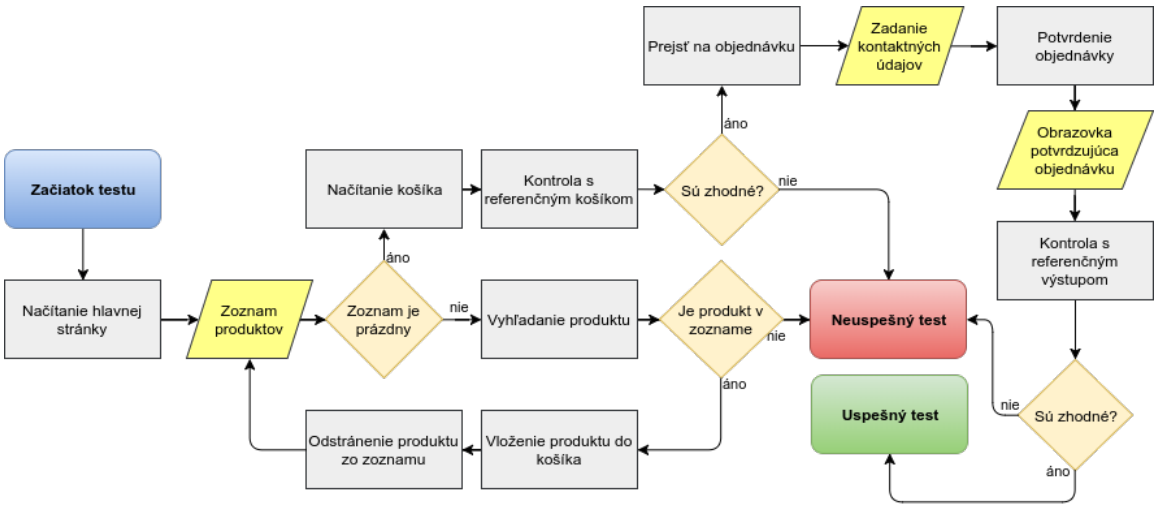
Kategória: Čierna skrinka
Dimenzia: Funkčnosť
Oblasť: Zoznam produktov a nákupný košík
Metrika: Porovnávanie obrázkov okna prehliadača na zhodu s referenčnými obrazmi



Obr. 4.2: Test vytvorenia objednávky (vlastné)

4.1.3 Tvorba objednávky viacerých produktov s vyhľadávaním

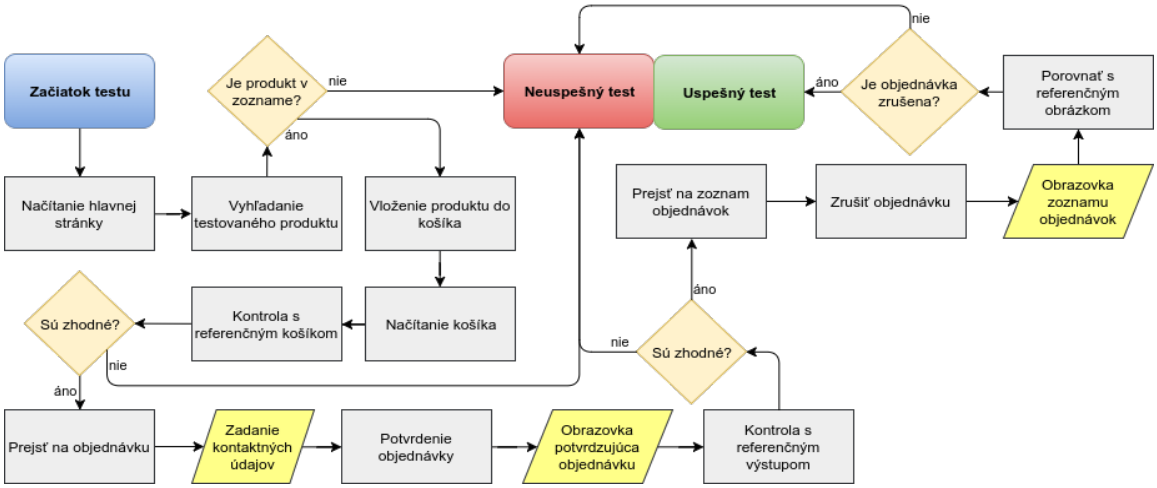
Kategória: Čierna skrinka
Dimenzia: Funkčnosť
Oblasť: Zoznam produktov, nákupný košík a vyhľadávanie
Metrika: Porovnávanie obrázkov okna prehliadača na zhodu s referenčnými obrazmi



Obr. 4.3: Test vyhľadávania a vytvorenia objednávky (vlastné)

4.1.4 Zrušenie objednávky

Kategória: Čierna skrinka
Dimenzia: Funkčnosť
Oblasť: Nákupný košík
Metrika: Porovnávanie určitej oblasti na zhodu s referenčnými obrazmi



Obr. 4.4: Test zrušenia objednávky (vlastné)

4.2 Aplikácia

Návrh architektúry bol vytvorený s ohľadom na hlavné ciele. Veľký ohľad sa bral hlavne na rozšíriteľnosť a to z dôvodu veľkého množstva kategórie testov, ktoré existujú a neboli spomenuté v tejto práci, ale môžu byť dodatočne implementované. Z tohto dôvodu bola vybraná 3 vrstva architektúra skladajúca sa z: webového rozhrania, servera a klienta.

Klient ako súčasť architektúry je skript napísaný v jazyku JavaScript, ktorý slúži ako zachytávač udalostí na webovej stránke a ich úprave pre **Server**. Skript musí byť vložený do webovej stránky pre ktorú chceme vytvoriť test. Toto riešenie by malo spoločne s kombináciou **webového rozhrania** zjednodušiť vytváranie testovacích scenárov pre testerov a tým zvýšiť používanie testov ako súčasť vývojového cyklu.

Webové rozhranie následne má slúžiť ako jednoduché užívateľské rozhranie pre editovanie, vytváranie, kontrolu a spúšťanie testov. Pre jeho implementáciu bude využitý tiež jazyk Javascript v podobe frameworkov **React** a **Semantic UI**.

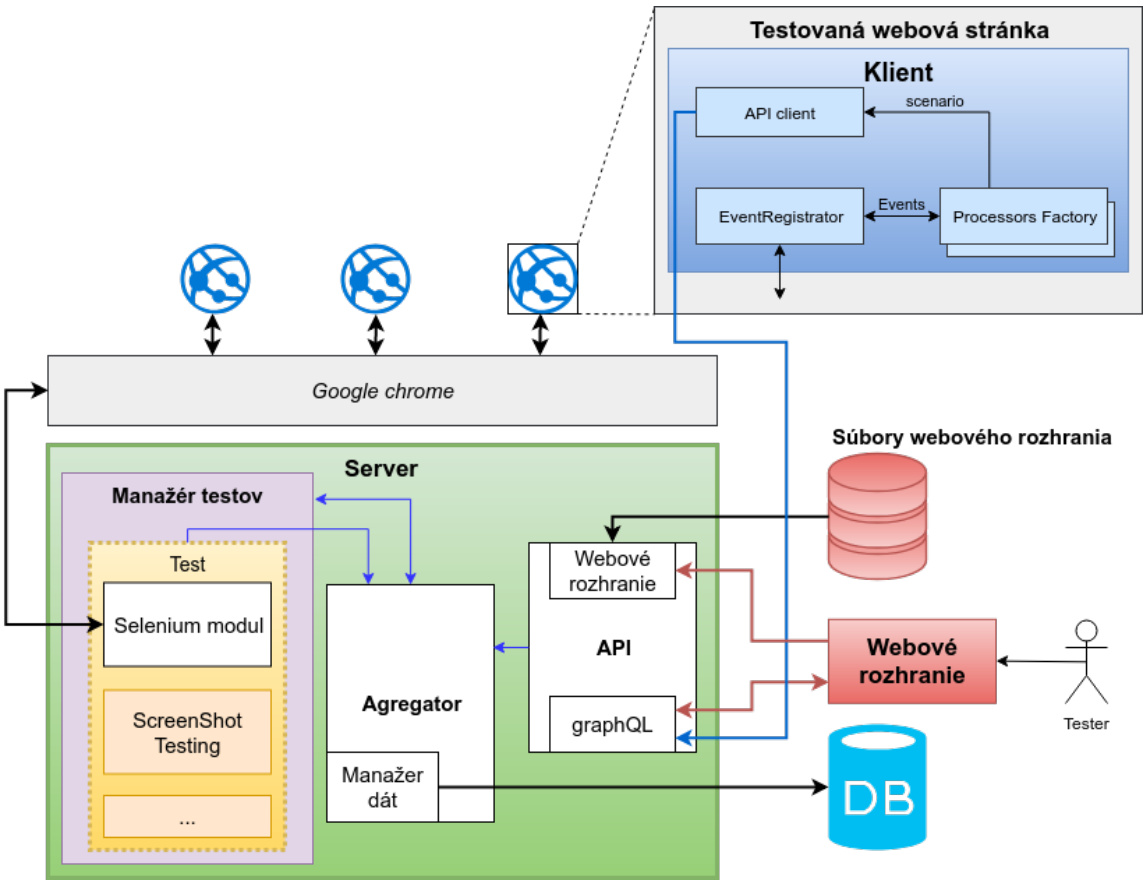
Následne o celú funkčnosť sa postará **Server**, čím bude možné v budúcnosti vytvoriť viac rôznych klientov alebo užívateľských rozhraní pre špeciálne potreby. Samotný server bude tiež modulárny, aby bolo možné ľahko pridávať novú funkcionalitu formou modulov. Kvôli tomuto účelu je ústredným členom celej serverovej časti takzvaný **Agregátor**, ktorého hlavná funkcia je preposielanie správ medzi jednotlivými modulmi.

K samotnému testovaniu následne bude využitá aplikácia **Selenium** [8] s ktorou bude možné komunikovať pomocou rovnako pomenovaného modulu **Selenium**. Ďalšou dôležitou súčasťou sú samotné dáta, ktoré musí server spravovať a ukladať.

Aby bola aplikácia menej závislá na voľbe úložiska dát je súčasťou takzvaný **Manažér dát**, ktorý má zabezpečiť transformáciou dát z úložiska do internej reprezentácie pre server a opačne.

Všetky tieto moduly sú dôležité pre fungovanie serverovej časti, ale hlavnú funkcionalitu budú zabezpečovať testovacie moduly a **Manažér testov**, ktoré by mali zabezpečiť samotné testy nad dátami získaných zo Selenia. **Manažér testov** má zároveň slúžiť ako správca testovacích scenárov pre zabezpečenie automatizácie testovania.

Posledným modulom je **Webový server**, ktorý môže fungovať len ako samostatné **GraphQL API** a umožní tak komunikáciu so serverovou časťou, s klientom alebo webovým rozhraním. Ale tiež môže slúžiť ako webový server pre prístup k webovému rozhraniu.



Obr. 4.5: Architektúra aplikácie (vlastné)

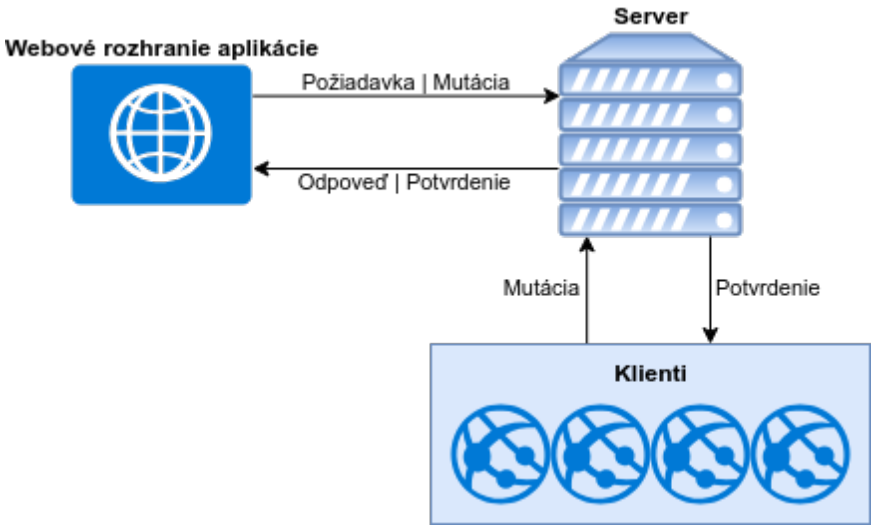
Kapitola 5

Implementácia

Implementácia bude vychádzať z predchádzajúcej kapitoly 4 Návrh. V tejto kapitole bude podrobnejšie popísaná implementácia jednotlivých častí aplikácie so zameraním na problémy, ktoré sa objavili počas vývoja.

5.1 Komunikácia

Jednotlivé časti aplikácie majú byť na seba nezávisle. Najdôležitejšia nezávislosť je medzi serverom a klientom, kde integrácia klienta do serveru by vyžadovala integrovanie samotnej testovanej webovej stránky do serverovej časti. Aby bola zabezpečená nezávislosť jednotlivých častí aplikácie, bol využitý komunikačný protokol graphQL pre komunikáciu medzi jednotlivými časťami. Komunikačný protokol graphQL je komunikačný protokol postavený nad aplikačným protokolom HTTP a bol vytvorený spoločnosťou Facebook. Bol vybraný ako náhrada za aplikačný protokol REST, nakoľko umožňuje lepšiu integráciu s knižnicou React pomocou voľne dostupného klienta Apollo.



Obr. 5.1: Návrh Komunikácie medzi časťami aplikácie (vlastné)

Centrálnym uzlom komunikácie je samotný Server, kde komunikáciu zabezpečuje rozšírenie pre webový server Flask spoločne s rozšírením graphene pre integráciu protokolu graphQL. K serveru sa pripájajú následne jednotlivé časti aplikácie vo forme dotaz-odpoveď.

Server následne spracuváva jednotlivé požiadavky od ostatných časti. Návrh je značne jednoduchý a skladá sa z množiny správ, ktoré má server implementované. Tieto správy sa na základe graphQL protokolu delia na dva typy (Požiadavky, Mutácie), ktoré môžu ovplyvňovať len stav samotného servera. Nie je teda možné pomocou webového rozhrania nijako priamo zasahovať do komunikácie alebo činnosti jednotlivých klientov.

5.1.1 Abstraktné dátové typy

Pre vhodnú reprezentáciu dát pri komunikácií boli navrhnuté abstraktné dátové typy, ktoré reprezentujú určitý aspekt aplikácie. Tieto abstraktné dátové typy vychádzajú z abstraktných dátových typov uvedených v sekcií 5.2 Databáza, časť 5.2.1 Abstraktné typy. V niektorých prípadoch sú ale modifikované pre použitie v jednotlivých častiach aplikácie.

Aplikácia

Oproti abstraktnému typu z podkapitoly 5.2 Databáza ide o veľmi zjednodušenú formu abstraktného typu aplikácie, ktorá bola navrhnutá pre potreby zobrazenia a vytvorenia testovanej webovej aplikácie s využitím webového rozhrania. Preto obsahuje len nasledujúce atribúty:

- **Identifikátor** (id) – Jedinečný identifikátor testovanej aplikácie
  - **Dátový typ** – Reťazec
  - **Vyžadovaný** – Áno
- **Doména** (domain) – Pomocný kľuč pre triedenie a riadenie aplikácií
  - **Dátový typ** – Reťazec
  - **Vyžadovaný** – V mutáciách
- **Dátum vytvorenia** (created) – Čas vytvorenia aplikácie
  - **Dátový typ** – Reťazec
  - **Vyžadovaný** – V mutáciách

Scenár

Scenár je abstraktný dátový typ, ktorý je zložený z dát obecných informácií uložených v riadiacom abstraktnom dátovom type Aplikácia a z agregovaných abstraktných dát typu Udalosť uložených v databáze. Pozostáva z nasledujúcich atribútov:

- **Identifikátor** (scenarioId) – Univerzálny jedinečný identifikátor (UUID) scenára v konkrétnej aplikácií
  - **Dátový typ** – Reťazec
  - **Vyžadovaný** – Nie
- **Názov** (name) – Užívateľský definovaný názov scenára
  - **Dátový typ** – Reťazec
  - **Vyžadovaný** – Nie

- **Identifikátor posledného testu** (lastTestId)
  - **Dátový typ** – Číslo so znamienkom (Int)
  - **Vyžadovaný** – Nie
- **Identifikátor aktuálneho regresného testu** (regressTestId)
  - **Dátový typ** – Číslo so znamienkom (Int)
  - **Vyžadovaný** – Nie
- **Stav testu** (state) – Reprezentuje aktuálny stav prebiehajúceho testu alebo výsledok posledného testu
  - **Dátový typ** – Číslo so znamienkom (Int)
  - **Vyžadovaný** – Nie
- **Počet udalostí** (events) – Agregované informácie o počte udalostí uložených v databáze s príslušným identifikátorom scenára.
  - **Dátový typ** – Číslo so znamienkom (Int)
  - **Vyžadovaný** – Nie

Udalosť

Tento dátový typ je najjednoduchší a zodpovedá priamo abstraktnému dátovému typu Udalosť z podkapitoly 5.2 Databáza.

Výsledok

Posledný abstraktný dátový typ, rovnako ako udalosť, odpovedá abstraktnému dátovému typu Výsledok z kapitoly Databáza, s tým, že je rozšírený o nasledné 3 atribúty:

- **Identifikátor** – Interný identifikátor výsledku v databáze.
  - **Dátový typ** – Reťazec
  - **Vyžadovaný** – Áno
- **Počet udalostí** – Agregované informácie o počte udalostí uložených v databáze s príslušným identifikátorom scenára.
  - **Dátový typ** – Číslo so znamienkom (Int)
  - **Vyžadovaný** – Nie
- **Stav testu** – Reprezentuje aktuálny stav prebiehajúceho testu alebo výsledok testu ku ktorému sa výsledok vzťahuje.
  - **Dátový typ** – Číslo so znamienkom (Int)
  - **Vyžadovaný** – Nie

Stav

Stav je súčasť iných komplexnejších abstraktných typov a vyjadruje primárne stav prebiehajúceho testu alebo výsledok ukončeného testu. Je definovaný pomocou čísla so znamienkom, kde jednotlivé hodnoty sú rozdelené do 2 skupín s rozsahom 100 položiek:

- **Výsledky testov** – Skupina v rozsahu čísel 0 až 99.
  0. **OK** – Test je úspešný.
  1. **COUNT_EVENTS** – Nevykonali sa všetky kroky testu.
  2. **FAILED** – Test je neúspešný.
- **Stav prebiehajúceho testu** – Skupina v rozsahu čísel 100 až 199.

  100. **INITIALIZE** – Inicializácia testu.
  101. **TESTING** – Replikácia udalostí.
  102. **ANALYZE** – Analýza snímkov ako súčasť regresného testovania.

Jednotlivé stavy môžu mať okrem číselnej a textovej formy aj grafickú reprezentáciu, ktorá sa využíva vo webovom rozhraní aplikácie. Nakoľko nie je možné zabezpečiť rovnakú implementáciu medzi jednotlivými časťami aplikácie z dôvodu použitia rôznych programovacích jazykov a prezentačnej forme stavov. Musia byť stavy implementované samostatne pre server a webové rozhranie aplikácie.

5.1.2 Správy

Pre komunikáciu medzi jednotlivými časťami aplikácie je navrhnutých 12 správ. Tieto správy sú rozdelené podľa graphQL na požiadavky a mutáciu, kde 11 správ je určených pre webové rozhranie a 1 správa je určená pre klientov. Všetky správy sú uvedené v tabuľke 5.1.

Požiadavky			
Názov	**Parametre**	**Návratová hodnota**	**Popis**
app	Identifikátor aplikácie	Zoznam aplikácií	Vráti základné informácie o všetkých alebo konkrétnej aplikácií
scenario	Identifikátor aplikácie	Zoznam scenárov	Vráti zoznam scenárov pre zadefinovanú aplikáciu
getResult	Identifikátor aplikácie, Identifikátor scenára, Zoznam identifikátorov testov (Nepovinný)	Zoznam výsledkov	Vráti výsledky jednotlivých krokov zadefinovaných testov
getResultAgg	Identifikátor aplikácie, Identifikátor scenára	Zoznam výsledkov	Vráti obecné výsledky všetkých testov v zadefinovanom scenári
getTest	Identifikátor aplikácie, Identifikátor scenára	Zoznam udalostí	Vráti všetky udalosti pre zadefinovanú kombináciu vstupov
generateClientUrl	Identifikátor aplikácie	URL pre stiahnutie klienta	Vygeneruje klientsky skript pre požadovanú aplikáciu
deleteApp	Identifikátor aplikácie	Správa o výsledku	Odstráni aplikáciu z databázy
Mutácie			
createApp	Aplikácia	Správa o výsledku	Vytvorí záznam o aplikácií v databáze
createEvent	Udalosť	Správa o výsledku	Vytvorí záznam o udalosti v databáze pre zadefinovanú aplikáciu
setScenarioName	Identifikátor aplikácie, Identifikátor scenára, Názov	Správa o výsledku	Priradí konkrétnemu scenáru užívateľské pomenovanie
setReressTest	Identifikátor aplikácie, Identifikátor scenára, Identifikátor testu	Správa o výsledku	Nastaví zadefinovaný test ako regresný pre nastavený scenár v aplikácií
runTest	Identifikátor aplikácie, Identifikátor scenára	Správa o spustení testu	Predá požiadavku manažérovi testov pre spustenie testu pre špecifikovaný scenár v aplikácií

Tabuľka 5.1: Zoznam implementovaných správ na servery (vlastné)

5.2 Databáza

Pre vývoj prototypu bolo dôležité zvoliť vhodné úložisko dát. Primárnym cieľom bolo ľahko rozširiteľné úložisko. Táto požiadavka je zároveň veľmi dôležitá aj pre samotného klienta, kde jednotlivé procesory môžu k jednotlivým udalostiam zaznamenávať rôzne informácie podľa potrieb testov. Teda nebolo možné využiť žiadnu voľne dostupnú SQL databázu ako je MySQL alebo PostgreSQL, nakoľko vyžadujú predom definovanú schému tabuliek.

Voľba databázy teda bola zo sekcie NoSQL databáz, kde výber bol z možností MongoDB, Elasticsearch a Redis. Nakoľko sa jednalo len o prototyp aplikácie, nebolo medzi databázami uskutočnené veľké porovnanie a rozhodovalo sa primárne na užívateľskej prívetivosti a mojich predchádzajúcich skúsenosti. Na základe tohto aspektu, bola zvolená databáza Elasticsearch, ktorá v kombinácií s vizualizačným nástrojom Kibana umožňuje ľahké testovanie dotazov, prehľadávanie a vizualizovanie dát. Zároveň Elasticsearch a Kibana majú repozitáre pre väčšinu známych linuxových distribúcií, ktoré zjednodušujú ich nasadenia.

5.2.1 Abstraktné typy

Abstraktné typy v databáze predstavujú štruktúru formátu JSON, v ktorom sú uložené perzistentné dáta aplikácie. Jednotlivé abstraktné typy nie sú ale statické a môžu byť dynamicky podľa potreby rozširované. Pre funkčnosť celej aplikácie sú navrhnuté 3 abstraktné typy, ktoré reprezentujú určité časti aplikácie.

Aplikácia (app)

Aplikácia je základný abstraktný typ. Jeho funkcia je hlavne riadiaca a centralizuje informácie o testovaných webových stránkach (interne pomenovanie: aplikácie), scenároch a výsledkoch.

Jedinečným identifikátorom dokumentu aplikácie je interný identifikátor elasticsearch databázy (_id), ktorý zodpovedá definovanému názvu aplikácie. Takéto riešenie zabraňuje duplicitným názvom aplikácií a umožňuje priamy prístupu k údajom aplikácie bez nutnosti vyhľadávacieho požiadavku do databázy.

Z obecných informácií pre aplikáciu následne JSON formát obsahuje dátum vytvorenia aplikácie a doménu. Tieto položky sú ako doplňujúce informácie pre ľahšie filtrovanie a hľadanie v prípade väčšieho počtu aplikácií.

Následne abstraktný typ Aplikácia obsahuje slovník objektov, ktoré reprezentujú globálne informácie o scenároch. Jednotlivé objekty sú prístupné pomocou UUID identifikátor scenára. Každý scenár obsahuje následujúce informácie:

- **Identifikátor posledného testu** (lastTestId) – Celočíselná premenná reprezentujúcu poradové číslo posledného testu. Pri každom novom teste je táto premenná inkrementovaná o číslo 1.
- **Identifikátor regresného testu** (regressTestId) – Premenná reprezentujúca číslo vykonaného testu, oproti ktorému budu testované všetky spustené testy.
- **Názov scenára** (name) – Užívateľské pomenovanie scenára pre lepšiu orientáciu medzi viacerými scenármi.
- **Testy** (tests) – Slovník obecných informácií o vykonaných testoch, kde jednotlivé testy sú dostupné pomocou číselného identifikátora testu.

  - **Identifikátor regresného testu** (regressTestId) – Permanentné uloženie identifikátora testu, ku ktorému bol daný test porovnávaný.
  - **Stav** (state) – Reprezentácia stavu a celkového výsledku testu.

Udalosť (event)

Abstraktný typ udalosť (event) reprezentuje primárne HTML udalosti, ktoré zachytáva klient na stránke a preposiela ich na server, kde sú združené podľa kombinovaného identifikátora. Udalosť by mala byť značne dynamická a jej obsah by sa mal odvíjať od potrieb testu, ale každá udalosť by mala obsahovať minimálne nasledujúce základe atribúty:

- **Identifikátor aplikácie** (appId) – Identifikuje konkrétnu testovanú webovú aplikáciu v systéme.
- **Identifikátor scenára** (scenarioId) – Identifikuje príslušnosť udalosti do konkrétneho scenára.
- **Lokátor** (locator) – Jedinečný identifikátor elementu stránky využívajúci CSS lokátor.
- **Cesta** (path) – Pole typov elementov určujúcich úroveň elementu v hierarchickej štruktúre stránky, na ktorej vznikla udalosť.
- **Typ** (type) – Typ udalosti, ktorá bola zachytená.
- **Adresa** (url) – URL adresa v prehliadači pri obsluhe udalosti.
- **Časové razítko** (timestamp) – Aktuálny čas v sekundách s milisekundami pri obsluhe udalosti. Využíva sa pre správne vytvorenie scenára na základe času vzniku jednotlivých udalosti.
- **Čas na stránke** (pageTime) – Uplynulý čas od načítania stránky po vytvorenie udalosti. Používa sa pre simulovanie užívateľskej činnosti pri vytváraní scenára.
- **Šírka okna** (screenX) – Rozmer okna prehliadača na X ose počas spracovania udalosti.
- **Výška okna** (screenY) – Výška okna prehliadača počas spracovania udalosti.
- **Obsah** (content) – Obsah elementov, do ktorých je možné zapísať textový vstup.

Výsledok (result)

Výsledok reprezentuje výsledok testu pre jednu konkrétnu udalosť. Výsledok je jednoznačne identifikovaný na základe času jeho vytvorenia, ktorý je uložený ako interný identifikátor dokumentu (_id). Nakoľko sú udalosti počas testu vykonávané v sekvenčnom poradí a po každom spracovaní udalosti je uložený výsledok, je možné po zoradení udalosti podľa časového razítka a výsledkov podľa identifikátora spárovať výsledky s odpovedajúcimi udalosťami. Výsledok ďalej obsahuje nasledujúce informácie:

- **Identifikátor testu** (testId) – Zoskupuje výsledky pod jeden konkrétny test.
- **Identifikátor regresného testu** (regressTestId) – Identifikuje test oproti ktorému bolo vykonané regresné testovanie.

- **Obrázok** (image) – relatívna cesta k obrazovému záznamu obsahu webovej stránky po vykonaní udalosti počas testu.
- **Skóre** (score) – Normovaný výsledok regresného testovania na základe porovnania rozdielov medzi obrazovými záznamami webových stránok aktuálneho a regresného testu.
- **Čas vykonania akcie** (performTime) – Potrebný čas pre replikáciu udalosti vo webovom prehliadači počas testovacieho procesu.

5.2.2 Ukladanie dát

Pre ukladanie dát bola navrhnutá 3-vrstva štruktúra registrov, do ktorých sa budú ukladať jednotlivé dáta. Tieto jednotlivé registre následne obsahujú dáta vo formáte abstraktných dátových typov.

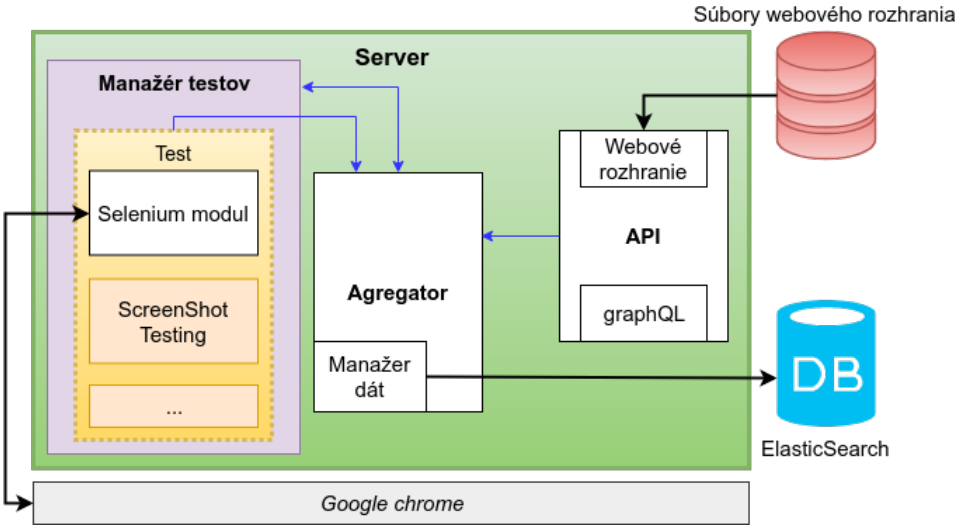
- **Riadiaci register** (manage) – Pozostáva z abstraktného typu Aplikácia (app), ktorý reprezentuje jednotlivé aplikácie alebo webové stránky. V databáze môže existovať len jeden takýto register.
- **Scenáre** (podľa názvu aplikácie) – Dokumenty typu scenáre pozostávajú z abstraktného typu Udalosť (event). V týchto registroch sú uložené všetky udalosti zachytené klientom, ktoré sa následne snažia napodobniť počas testov.
- **Výsledky** (result-názov_aplikácie-uuid_scenára) – Pre každý scenár existuje následne samostatný register, kde sú uložené jednotlivé výsledky testov ako abstraktný typ Výsledok (result).

5.3 Server

Najdôležitejšou časťou celej aplikácie je samotný server, ktorý zabezpečuje samotné testovanie a správu testovaných aplikácií. Celý server je implementovaný pomocou jazyka Python s využitím externých knižníc. Server rovnako ako celá aplikácia je modulárna, kde môžeme rozoznávať 4 základné moduly.

- **Hlavný program** – Zaobstaráva spustenie jednotlivých modulov a načítanie konfigurácie.
- **Aggregátor** – Centrálny prvok, ktorý zabezpečuje komunikáciu medzi ostatnými modulmi.
- **Manažér testov** – Riadenie spúšťania jednotlivých testov.
- **API** – Webový server s implementáciou graphQL protokolu pre komunikáciu s inými časťami aplikácie.

Pre celkovú funkčnosť a ľahké rozšírenie funkčnosti aplikácie jednotlivé 4 hlavné moduly obsahujú ďalšie moduly. Integrovaním modulov do modulov sa vytvára určitá hierarchická štruktúra, ktorú je možné vidieť na obrázku 5.2. Hlavné moduly ako súčasť aplikácie bežia ako samostatné vlákna s využitím knižnice Thread. Ostatné moduly môžu a nemusia bežať ako vlákna. Z ostatných modulov pre zabezpečenie testovania viacerých stránok súbežne bežia v samostatných vláknach len samotné testy reprezentované triedou Test.



Obr. 5.2: Bloková architektúra servera (vlastné)

5.3.1 ZeroMQ

Jazyk Python implementuje primárne komunikáciu medzi objektami s využitím referencie. Pre priblíženie aplikácie k distribuovanému systému bol navrhnutý interný komunikačný protokol medzi jednotlivými modulmi. Základom tohto komunikačného protokolu je protokol ZeroMQ, ktorý je implementovaný pomocou knižnice pyzmq a zapuzdrený v dvoch triedach pre jednoduchšie použitie ako súčasť servera. Tieto triedy v aktuálnej implementácií využívajú internú procesovú transportnú vrstvu (in-process), kde správy sú predávané pomocou zdieľanej časti pamäte [18].

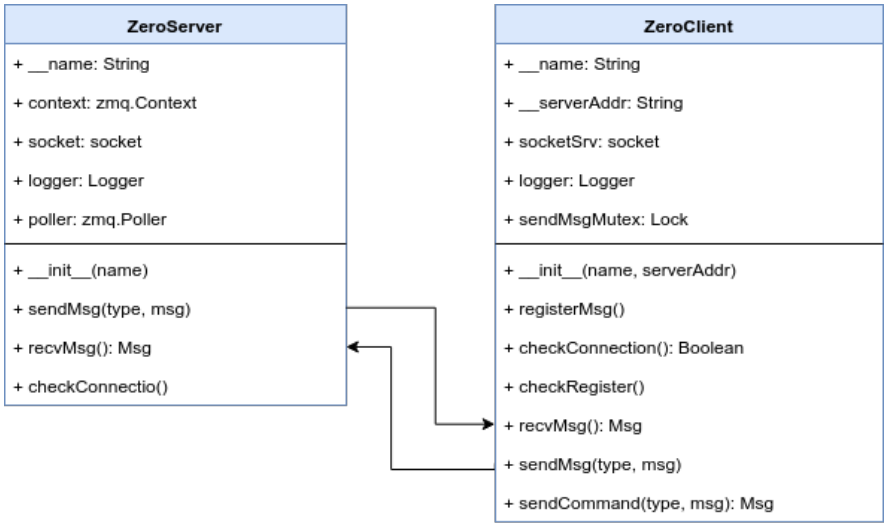
Pre posielanie správ sa následne používa nadstavba pre odosielanie objektov jazyka Python s využitím metód send_pyobj a recv_pyobj. Tieto metódy integrujú prevod objektu do správy a naspäť, čím zjednodušujú implementáciu celej komunikácie medzi modulmi.

Dôležitým aspektom komunikácie sú takzvané vzory správ, ktoré určujú spôsob nadviazania komunikácie a posielanie správ. V triedach sa využíva klasická architektúra Klient/Server, ktorá je reprezentovaná vzorom správ Dotaz/Odpoveď. Pre komunikáciu sa používajú rovnako ako v HTTP protokole jednoduché správy, ktoré nezachovávajú žiaden stav.

ZeroServer

Je postavený nad vzorom správy Odpoveď a v štandardnej architektúre odpovedá serveru. Každý server čaká na dotaz od klienta, ktorý je spracovaný a následne je na neho odpovedané. Nakoľko ZeroMQ negeneruje klientom jedinečné identifikátory ako napríklad BSD sokety, sú jednotlivé správy spracované sekvenčne.

Základné funkcie servera reprezentujú metódy sendMsg a recvMsg, ktoré majú podobnú implementáciu aj u klienta. Na servery sa ale využíva neblokujúce prijímanie správ pomocou časového limitu. Pre kontrolu nových správ je použitý dvoj-sekundový interval po ktorom je generovaná výnimka typu UserWarning, ktorá by mala byť zachytená v objekte, kde je vytvorená inštancia triedy ZeroServer. Táto vlastnosť bola implementovaná pre možnosť bezpečného ukončenia vlákna, kde vlákno bolo blokované pokiaľ nedostalo na konci nejakú správu.



Obr. 5.3: Triedy reprezentujúce server a klienta (vlastné)

ZeroClient

Implementácia klienta je podobná ako u servera. Klient pracuje s návrhovým vzorom typu Požiadavka, kde po odoslaní požiadavky vždy očakáva nejakú odpoveď. Klient rovnako ako v štandardnej architektúre Klient/Server zahajuje komunikáciu. Jeden klient môže komunikovať len s jedným serverom. Pre komunikáciu s viacerými servermi je potrebné vytvoriť inštancie viacerých klientov.

Oproti serveru je na strane klienta rozšírená funkcionalita pomocou metódy sendCommand, ktorá kombinuje volanie metód sendMsg a recvMsg, čím znovu zjednodušuje použitie klienta v jednotlivých moduloch. Pred zavolaním metódy sendMsg je v metóde sendCommand požiadavke pre uzamknutie mutexu sendMsgMutex. Toto opatrenie vyplýva z obmedzenia zeroMQ v aktuálnej implementácií, ktoré udržuje interný stav komunikácie na strane servera a v prípade odoslania dvoch požiadaviek, bez zaslania odpovede sa považuje za chybu v komunikácií a je vygenerovaná výnimka. Mutex teda zabraňuje odoslaniu ďalšej požiadavky, pokiaľ neprišla odpoveď na predošlú požiadavku.

Ďalším rozšírením klienta je metóda registerMsg a checkRegister, ktoré sa používajú pre oznámenie Agregátoru, že objekt, kde je inštancia triedy ZeroClient, obsahuje aj inštanciu triedy ZeroServer. Pre túto funkcionalitu server aj klient implementujú metódu checkConnection, ktorá generuje správu typu ping.

Správy

Pre komunikáciu medzi jednotlivými inštanciami klienta a servera bol navrhnutý jednoduchý formát správ, ktorý je vyjadrený objektom s nasledujúcimi atribútmi:

- **Odosielateľ** (from) – Adresa odosielateľa, v aktuálnej implementácií názov nadradeného objektu.
- **Príjemca** (to) – Nepovinný údaj pre kontrolu, že správa bola doručená požadovanému serveru alebo klientovi.
- **Typ** (type) – Typ alebo názov správy, reprezentuje ako sa správa použije.

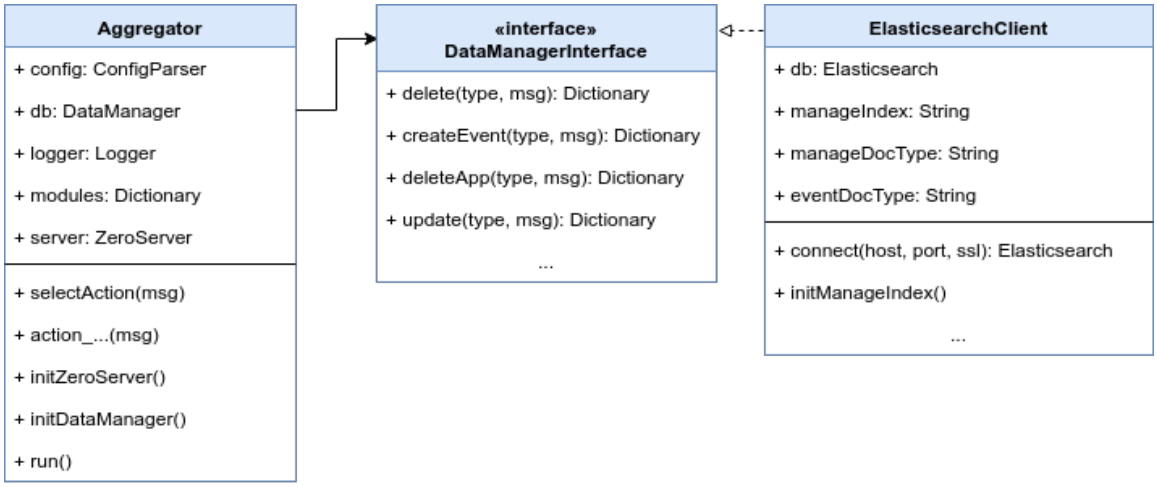
- **Správa** (msg) – Telo správy reprezentuje slovník podľa typu správy.

Nastavenie odosielateľa a príjemcu je zabezpečené v metóde sendMsg na základe parametrov konštruktora a tieto údaje by sa nemali za behu upravovať. Zároveň pre jednotlivé typy správ nie je počas komunikácie žiadna kontrola obsahu správy. Preto je na programátorovi, aby zabezpečil korektný obsah správy pri odosielaní, alebo následnú kontrolu po prijatí správy. V prípade zlyhania spracovania požiadavky v objektoch s inštanciou servera sú zachytávané výnimky a je odoslané odpoveď o neúspechu. Aby bolo možné rozoznať či požiadavka bola vykonaná korektne je požadovaná odpoveď zabalená v časti Správa pomocou slovníka s nasledujúcimi atribútmi:

- **Stav** (status) – Boolovská hodnota, ktorá identifikuje či došlo počas spracovania požiadavky k problémom. Na základe jeho hodnoty sa určuje jeden z nasledujúcich atribútov.
- **Chyba** (error) – Informácie o chybe. Väčšinou text výnimky.
- **Dáta** (data) – Výsledné dáta, ak boli požadované ako súčasť požiadavky.

Objekty v servery ktoré obsahujú inštancie ZeroServera spracuvávajú typy správ pomocou metód, ktoré odpovedajú hodnote typu správy s prefixom action_. Prefix je pridaný z dôvodu zvýšenia čitateľnosti kódu a primárne za účelom obmedzenia volania metód triedy, čím sa zvýši bezpečnosť celého kódu pri zachovaní možnosti ľahkého rozšírenia o nové správy.

5.3.2 Agregátor



Obr. 5.4: Agregátor (vlastné)

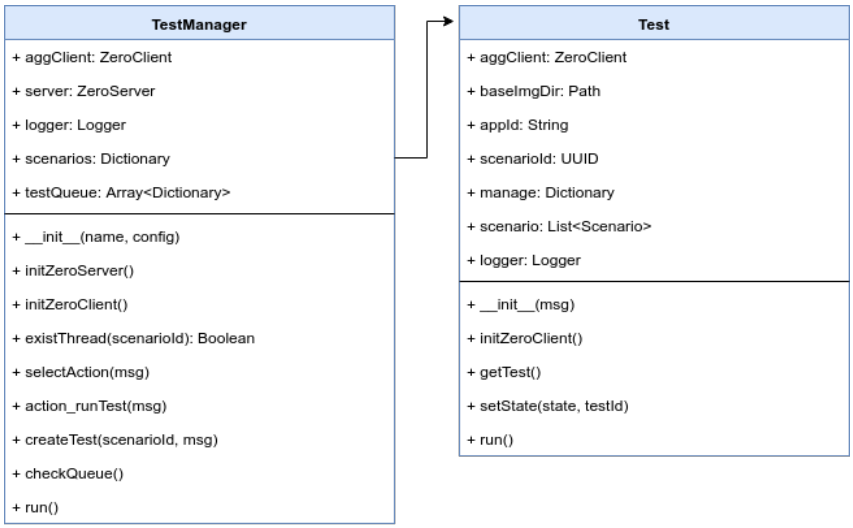
Agregátor ako ústredný člen sa primárne stará o smerovanie správ medzi jednotlivými modulmi. Pre zabezpečenie tejto funkcionality, Agregátor musí mať informácie o všetkých ostatných ZeroServerov a pre každý musí vytvoriť spojenie pomocou inštancie triedy ZeroClient.

Agregátor zároveň obsahuje inštanciu triedy manažéra Dát, ktorý je automatický generovaný pomocou rozhrania DataManagerInterface a príslušnou realizáciou daného rozhrania. V prípade implementácie aplikácie je použité pre komunikáciu s databázou Elasticsearch knižnica elasticsearch, ktorá je pre potreby rozhrania zapuzdrená do triedy ElasticsearchClient.

Celková funkčnosť ja zabezpečená v metóde run, kde sa v nekonečnom cykle čaká na prijatie správy od klienta, ktorá je následne predaná metóde selectAction. V metóde selectAction na základe typu správy je volaná príslušná metóda agregátoru, ktorá by mala implementovať obsluhu pre daný typ správy. Ak nie je nájdená príslušná metóda alebo počas obsluhy je vyvolaná výnimky, bude generovaná chybová správa klientovi, ktorý správu odoslal. Po úspešnom spracovaní správy alebo výnimky sa agregátor znovu vráti do metódy run, kde bude čakať na správy od klientov.

Aby nebola aplikácia závislá na 1 type úložiska, je práca s dátami riešená pomocou rozhrania DataManagerInterface, ktorý dedí triedu ABC. Použitie triedy ABC umožňuje v jazyku Python definovať abstraktné metódy a v prípade chýbajúcej realizácie je pri spustení aplikácie generovaná výnimka. Takéto riešenie je bezpečnejšie oproti čistému Pythonu, ktorý by vygeneroval výnimku až pri pokuse o volanie neexistujúcej metódy.

5.3.3 Manažér testov



Obr. 5.5: Manažér testov (vlastné)

Manažér testov sa skladá celkovo zo 4 tried, kde triedy TestManager a Test zabezpečujú riadiacu funkciu samotného testu a komunikáciu s inými modulmi aplikácie. Triedy seleniumClient a analyzeScreenshot následne riešia vykonávanie a analýzu výsledkov testov.

Riadenie testov

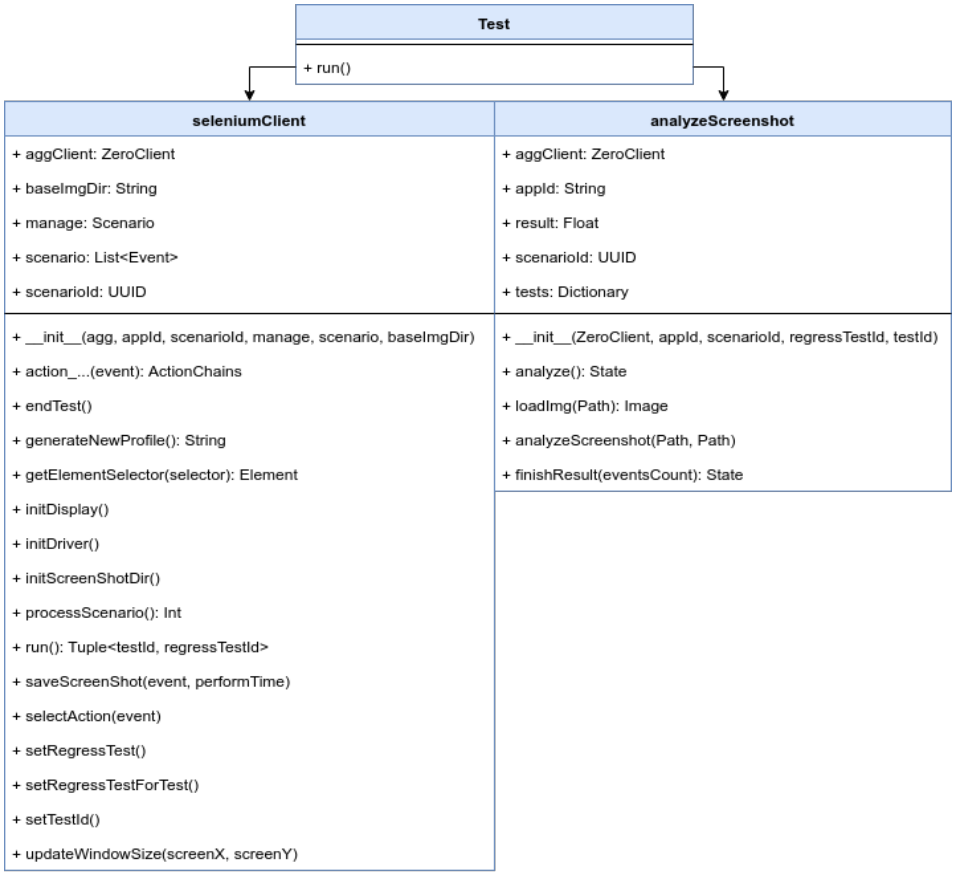
Manažer testov v základe funguje ako agregátor, ale v prípade spracovania správy alebo výnimky o žiadnej správe, dochádza ku kontrole fronty testov. Fronta testov obsahuje jednotlivé správy s požiadavkami pre vykonanie testu, ktoré boli prijaté počas prebiehajúceho testu pre rovnakú aplikáciu a scenár. Pre kontrolu prebiehajúceho testu, ktorý zabezpečuje trieda Test, ktorá beží ako samostatné vlákno, je dané vlákno uložené v premennej scenára a je kontrolované pri prijatí novej požiadavky. V prípade, že je vlákno živé zaradí sa požiadavka do fronty.

Pri vytváraní nového testu, je požiadavka predaná konštruktoru triedy Test, ktorý následne na základe správy bude kontaktovať agregátor pre zaslanie potrebných informácií k testu ako sú:

- **scenár**
- **obecné informácie o scenári**
  - **identifikátor posledného testu**
  - **identifikátor regresného testu**

Následne Test aktualizuje stav testu podľa potreby a zabezpečuje inštanciu tried seleniumClient a analyzeScreenshot a príslušných metód daných tried.

Testovanie



Obr. 5.6: Testovacie moduly (vlastné)

Hlavným testovacím nástrojom je nástroj Selenium, ktorý dokáže kontrolovať akcie vo webovom prehliadači. Pre implementáciu Selenium je použitá Python knižnica selenium spoločne s prehliadačom google chrome. Aby bolo možné prehliadač google chrome ovládať pomocou knižnice selenium je potrebné stiahnuť program chromedrive. Implementácia selenium modulu sa napokon stará o replikáciu udalostí, ktoré sú zachytené pomocou klienta. Ako súčasť testu sa tiež zaznamenáva doba vykonania jednotlivých udalostí, ktoré sú následne reprezentované pomocou 5.5.2 Grafu.

Okrem replikácie vykonania jednotlivých udalostí je snaha napodobiť aj chovanie užívateľa na stránke. Čas vykonania jednotlivých udalostí sa sčítava a následne sa porovnáva

s časom na stránke, kedy bola vykonaná nasledujúca udalosť. V prípade, že udalosť bola vykonaná neskôr ako je aktuálny súčet časov vykonaných udalostí, dôjde k pozastaveniu testovacieho procesu po dobu rozdielu času strávenom na stránke pre aktuálnu udalosť a súčtu časov potrebných pre vykonanie predchádzajúcich udalostí. Pri tomto type testov sa nepočíta s časovými požiadavkami pre vykonanie kódu. Pri návrhu bola táto skutočnosť známa a bola snaha umiestniť kódy pre výpočty času vykonania udalostí a pozastavenie procesu testovania tak, aby čo najmenej na nich vplývala réžia pre vykonanie kódu.

Aby bola zabezpečená funkčnosť prehliadača google chrome je potrebné, aby server disponoval grafickým výstupom. Nakoľko má serverová časť aplikácie primárne bežať na servery, kde vo väčšine prípadov chýba grafický výstup je potrebné použiť programovú náhradu. Za týmto účelom je potrebné nainštalovať program Xvfb, ktorý vytvorí virtuálny zobrazovací server v pamäti počítača. Pre implementáciu podpory Xvfb je v kóde použitá následne knižnica PyVirtualDisplay.

Po vykonaní replikácie udalostí nasleduje vyhodnotenie uložených obrázkov okna prehliadača, ktoré boli zhotovené po replikácií každej udalosti. Vyhodnotenie obrázkov prehliadača prebieha na základe staršieho testu, ktorý je manuálne nastavený a považuje sa za takzvaný regresný test. Pre porovnanie jednotlivých obrázkov sa používa metóda SSIM, ktorá berie do úvahy ľudské vnímanie scény. Tento algoritmus je implementovaný pomocou knižnice skimage a je definovaný vzorcom:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Obr. 5.7: SSIM vzorec (prevzaté [2])

5.3.4 API

API je reprezentované inštanciou triedy Flask a implementuje správy popísane v sekcií 5.1 Komunikácia, podľa potrieb knižnice graphene. Zároveň obsluha každej správy dokáže pristupovať ku globálnemu kontextu knižnice graphene, kde je pri inicializácií vytvorená inštancia triedy ZeroClient, pre potreby komunikácie s Agregátorom.

API okrem rozhrania pre komunikáciu medzi jednotlivými časťami aplikácie zabezpečuje prístup k obrázkom okna prehliadača, ktoré boli vytvorené počas procesu testovania. API rozlišuje následujúce url:

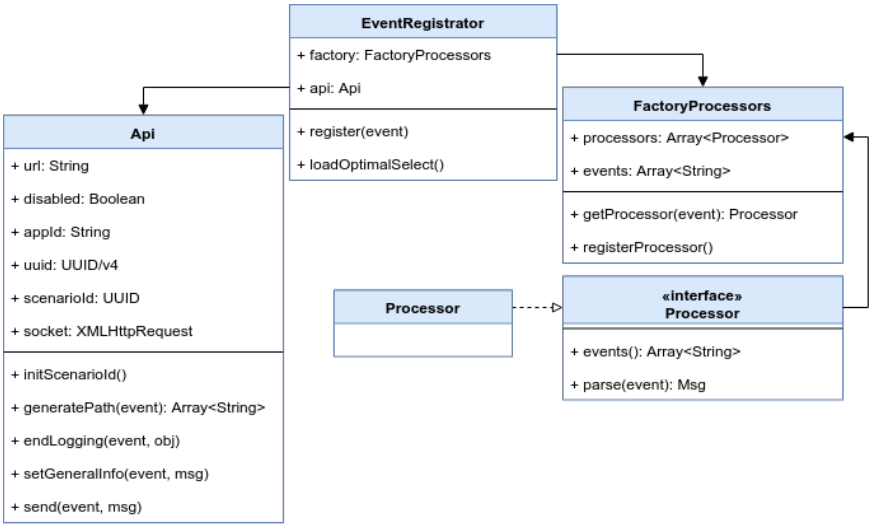
- **/client/** – Prístup k úložisku vygenerovaných klientov.
- **/graphqlTesting** – GraphQL API pre komunikáciu s inými časťami aplikácie.
- **/screenshot/** – Prístup k obrázkom vygenerovaných počas testovacieho procesu.
- **/static/*** (nepovinné) – Statické súbory webového rozhrania.
- **/*** (nepovinné) – Súbory webového rozhrania.

5.4 Klient

Pri implementácií klienta bol kladený dôraz na jednoduchosť a rozšíriteľnosť. Z dôvodu jednoduchosti bolo dôležité minimalizovať použitie externých knižníc a snažiť sa vytvoriť

implementáciu s čistým JavaScript. Dôvodom nevyužitia externých knižníc je primárne riziko narušenia funkčnosti samotnej testovanej webovej stránky a preto sa pri implementácií používajú len 2 externé knižnice:

- **optimal select** – Knižnica pre vytváranie CSS selektorov pre identifikáciu elementov na webovej stránke.
- **uuid** – Generovanie UUID podľa RFC4112 [13].



Obr. 5.8: Architektúra klienta (vlastné)

Celkový návrh klienta je jednoduchý a pozostáva z 3 základných tried, ktoré zabezpečujú základnú funkčnosť samotného klienta. Dôležitou časťou sú samotné procesory, ktoré je možné použiť pre rozšírenie spracovanie jednotlivých udalosti a generovanie správ pre odoslanie na server. Nakoľko je klient rozdelený na niekoľko súborov, využíva pre zostavenie finálneho klienta nástroj webpack, ktorý spojí jednotlivé súbory do jedného minimalizovaného JavaScript súboru.

EventRegistrator

Základná trieda, ktorá zapuzdruje celkovú funkčnosť klienta a ktorá je inicializovaná do globálnej premennej na konci súboru. Takéto riešenie umožňuje ľahké využitie klienta, kde stačí vložiť príslušný skript do stránky a nie je potrebná žiadna ďalšia interakcia. Následne sa ako súčasť inicializácie triedy prevedú tieto úkony:

1. **Inicializácia ďalších tried**
2. **Vloženie referencie na knižnicu optimal select, do hlavičky stránky (funkcia loadOptimalSelect)**
3. **Namapovanie funkcií pre zachytávanie udalosti**

Posledný úkon je najdôležitejší a rieši sa pre každý procesor a typ udalosti separátne. Takéto riešenie umožňuje jednoduché priradenie jedného procesora pre udalosť alebo N

procesorov pre jednú udalosť alebo skupinu udalosti. Táto vlastnosť môže byť využitá pri návrhu komplexnejších testovacích scenárov.

Pre zabezpečenie obsluhy udalosti sa využíva funkcionalita prehliadačov voľne preložená ako zachytávač udalostí, ktorý je nadviazaný na HTML objekt dokument. K tomuto účelu sa používa metóda objektu dokument s názvom addEventListener. Použitie tejto metódy je vhodnejšie a jednoduchšie pre inicializáciu. Iná možnosť bola vytvoriť zachytávač udalosti priamo na konkrétnych elementoch, ale nakoľko má byť použitie nezávislé na testovanej stránke, bolo by nutné inicializovať zachytávač udalosti pre každý element testovanej stránky. To by bolo značne náročné pri zložitých webových stránkach a mohlo by dôjsť k spomaleniu načítania samotnej testovanej stránky. Zároveň by vznikol značný problém s dynamicky pridávanými elementami, ktoré by bolo nutné sledovať a dodatočne im pridať zachytávače udalosti.

```
1 document.addEventListener(in DOMString type,
2         in EventListener listener,
3         in boolean useCapture);
```

Výpis 5.1: Rozhranie metódy addEventListener (prevzaté [15])

Pri zachytávaní udalosti sa následne využíva režim "Zachytenie", ktorý je dôležitý pre správnu funkčnosť skriptu. S využitím tohto režimu sú funkcie pre spracovanie udalosti vykonané pred zavolaním funkcií definovaných priamo na elemente. Čo je dôležité, v prípade elementov, ktoré po vykonaní udalosti sú odstránené z objektového modelu dokumentu (DOM). Bez využitia režimu "Zachytenie"dochádzalo k neúspešnému získaniu selektora elementu v knižnici optimal select, nakoľko element uložený v atribúte cieľ ako súčasť udalosti už neexistoval.

FactoryProcessors a Procesory

Návrhový vzor továreň pre procesory bol zvolený pre ľahšiu implementáciu nových procesorov a zabezpečenie jednotného rozhrania. Samotné procesory následne implementujú zoznam udalostí, ktoré má procesor obsluhovať. Okrem toho musí každý procesor obsahovať metódu parse, ktorá ma generovať správu o udalosti pre server.

API

Ako vyplýva z názvu primárna funkcia API je zabezpečenie komunikácie so serverom. Nakoľko má byť klient jednoduchý, bez využitia veľkého počtu knižníc a server komunikuje s využitím graphQL, nebola použitá žiadna špeciálna knižnica pre komunikáciu. Pre komunikáciu sa používa AJAX, kde je definovaná len jedna správa pre nahranie udalosti vo formáte graphQL správ.

Sekundárnou funkcionalitou API je správa samotného testovacieho procesu. API pomocou knižnice uuid generuje pre každý test UUID kód, ktorý má odstránené znaky -. Pre uloženie UUID počas celého procesu nahrávania udalosti sa používa moderný typ Web úložiska v prehliadačoch. Aby nedochádzalo k problémom s opakovaným použitím existujúceho UUID je pre uloženie použitý typ úložiska "Zasadanie"(angl. Session), ktoré má platnosť do uzavretia stránky, ale zároveň jeho platnosť sa vzťahuje na celú doménu. Pre manuálne ukončenie nahrávania testovacieho scenára je implementovaná aj udalosť pre klávesovú skratku CTRL+Z, ktorá zmaže UUID z úložiska a prepne API to neaktívneho režimu, kedy nie sú správy o udalostiach ďalej posielané na server.

API zároveň slúži ako centrálne miesto, ktorým sú spracované všetky udalosti. Preto API slúži ako základný procesor, ktorý nie je ale viazaný na žiaden typ udalosti. Pred samotným odoslaním správy z procesora API modifikuje jednotlivé správy a rozšíri ich o potrebné atribúty ako je aktuálna URL v prehliadači, rozmery okná, UUID a ďalšie. API by malo teda pridávať atribúty, ktoré sú spoločné pre všetky typy udalostí, alebo ktoré sú dôležité pre samotný testovací proces. Takéto riešenie znižuje značne duplicitu a zložitosť zdrojového kódu samotného klienta.

5.5 Webové rozhranie

Hlavným účelom webového rozhrania bolo vytvoriť prívetivé užívateľské rozhranie pre ovládanie aplikácie. Webové rozhranie vytvára prívetivejší spôsob volania API príkazov pomocou grafických prvkov. Webové rozhranie bude v tejto sekcii popísané z grafického pohľadu v sekciách 5.5.1 Grafický dizajn až 5.5.4 Obsah webovej stránky. V sekcií 5.5.5 Programová štruktúra sa zameriam na určitú časť funkčného aspektu webového rozhrania.

5.5.1 Grafický dizajn

Dizajn webovej stránky sa odvíjal od nasledujúcich pravidiel:

- **Jednoduchosť a použiteľnosť** – Prostredie má byť intuitívne a ľahko ovládateľné. Stránka ma pôsobiť minimalisticky a počet ovládacích prvkov nesmie byť veľký, ak to nie je nutné.
- **Prívetivosť** – Akcie užívateľa by mali mať vizuálnu odozvu.
- **Konzistentnosť** – Grafický dizajn ma pôsobiť ucelene, bez zbytočného používania veľkého počtu grafických prvkov.
- **Vhodný na prácu v noci** – Za účelom práce v tmavých priestoroch alebo v nočných hodinách, musí byť grafický dizajn navrhnutý tak, aby minimalizoval vyžarované svetlo na zobrazovacom zariadení.

Na základe týchto pravidiel je navrhnuté celé užívateľské rozhranie. Výsledné užívateľské rozhranie teda vo väčšine prípadov je pokryté minimálne z 80% tmavou farbou, aby bolo dosiahnuté vhodné pracovné prostredie pre prácu v noci. Za týmto účelom pozadie stránky je tvorené tmavými odtieňmi sivej farby. Následne ovládacie prvky sú farebne odlišné, aby vznikol kontrast medzi prvkami a pozadím, čo umožní ľahšie rozlíšenie ovládacích prvkov. Jednotlivé grafické prvky sú väčšinou postavené nad grafickou knižnicou Semantic UI, ktorá zabezpečuje konzistenciu grafického prostredia. Pre zlepšenie práce s užívateľským rozhraním boli tiež pridané animácie pre jednotlivé akcie užívateľa, čo by malo zvýšiť prívetivosť stránky a informovanosť o aktuálnom stave aplikácie. Aby bola zabezpečená použiteľnosť a jednoduchosť, celá stránka pozostáva zo šiestich hlavných prvkov, ktoré sú následne modifikované podľa potreby obsahu.

5.5.2 Prvky stránky

Prvky stránky môžeme rozdeliť na:

- **Pomocné**
- **Funkčné**

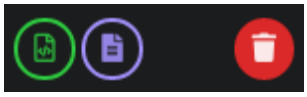
Pomocné prvky slúžia ako kontajnery alebo doplnkové informácie, ktoré zaobaľujú funkčné prvky, upravujú vzhľad stránky, alebo zvyšujú použiteľnosť stránky. Medzi tieto prvky môžeme zaradiť jednoduché elementy div, span, tlačidla a iné. Väčšina pomocných prvkov stránky je použitá z knižnice Semantic UI. Knižnica Semantic UI bola vybraná ako alternatívna knižnica ku knižnici Bootstrap, ktorá bola primárna voľba na začiatku vývoja. Ale pri vývoji dochádzalo k problémom počas prekladu kvôli nedostatočnej integrácií s knižnicou React a podporou štandardu ES6.

Funkčné prvky sprostredkuvávajú určitú funkčnosť samotnej aplikácie. V aktuálnom stave implementácie sú využité len dva funkčné prvky:

- **Porovnávač obrázkov**
- **Graf**

Tlačidla

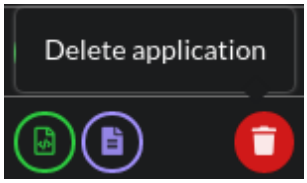
Tlačidlá sú najdôležitejšie pomocné prvky, nakoľko majú aj funkčnú časť. Tlačidla sú prezentované väčšinou pomocou elementu <Button> z knižnice Semantic UI, kde prezentácia tlačidla je vo väčšine prípadoch len grafická pomocou elementu <Icon>. Aby bola zlepšená prívetivosť webovej stránky, sú tlačidlá rozšírené o takzvané vyskakujúce okná. Okrem klasických tlačidiel sú používane aj tlačidla v grafickej forme prepínača.



Obr. 5.9: Dizajn tlačidiel (vlastné)

Vyskakujúce okná

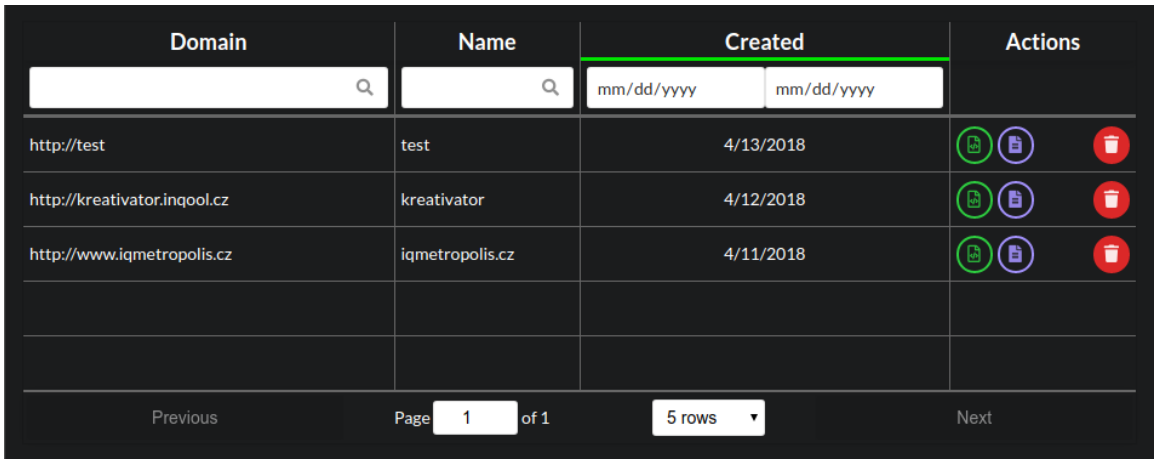
Tieto okná sú primárne nositeľom dodatočnej textovej informácie k tlačidlám. Vyskakujúce okná sú realizované pomocou elementu <Popup>. Element <Popup> slúži ako kontajner, ktorý zaobaľuje požadovaný element pri ktorom sa má zobraziť dodatočná informácia, ak myš je nad zaobaleným elementom.



Obr. 5.10: Vyskakujúce okno (vlastné)

Zoznam

Zoznam je jedným z kľúčových prvkov pre zobrazovanie informácií. Pre jeho implementáciu bola použitá externá knižnica React Table, ktorá bola vybraná na základe ľahkej modifikovateľnosti, integrovanej podpore stránok, vyhľadávania a radenia.



Obr. 5.11: React Table (vlastné)

Pre dodržanie pravidiel zo sekcie 6.1 Grafický dizajn boli vytvorené vlastné vstupné elementy pre vyhľadávanie pomocou elementov <Input> z knižnice Semantic UI v kombinácií s elementom <Icon>, ktorý reprezentoval ikonu lupy, čím by sa malo dosiahnuť lepšej použiteľnosti.

Záložky

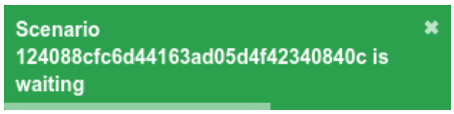
Záložky sú implementované s využitím elementu <Tab> z knižnice Semantic UI a skladajú sa z dvoch časti:

- Panel záložiek – Ovládací prvok pre prepísanie obsahu. Obsahuje všetky aktívne záložky.
- Obsah záložky – Zobrazuje obsah aktívnej záložky.

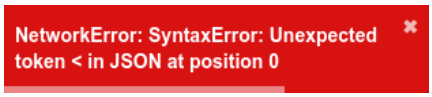
Obr. 5.12: Panel záložiek s obsahom (vlastné)

Notifikácie

Za účelom zvýšenia použiteľnosti a interaktivity webovej stránky je integrovaná podpora notifikácií s využitím knižnice React Toastify, ktorá rovnako ako React Table umožňuje jednoduchú modifikovateľnosť a použiteľnosť. V projekte sú navrhnuté dva typy notifikácie pre úspešné potvrdenie akcie a pre informovanie o chybe. Každá notifikácia má zároveň časové obmedzenie zobrazenia na 8 sekúnd.

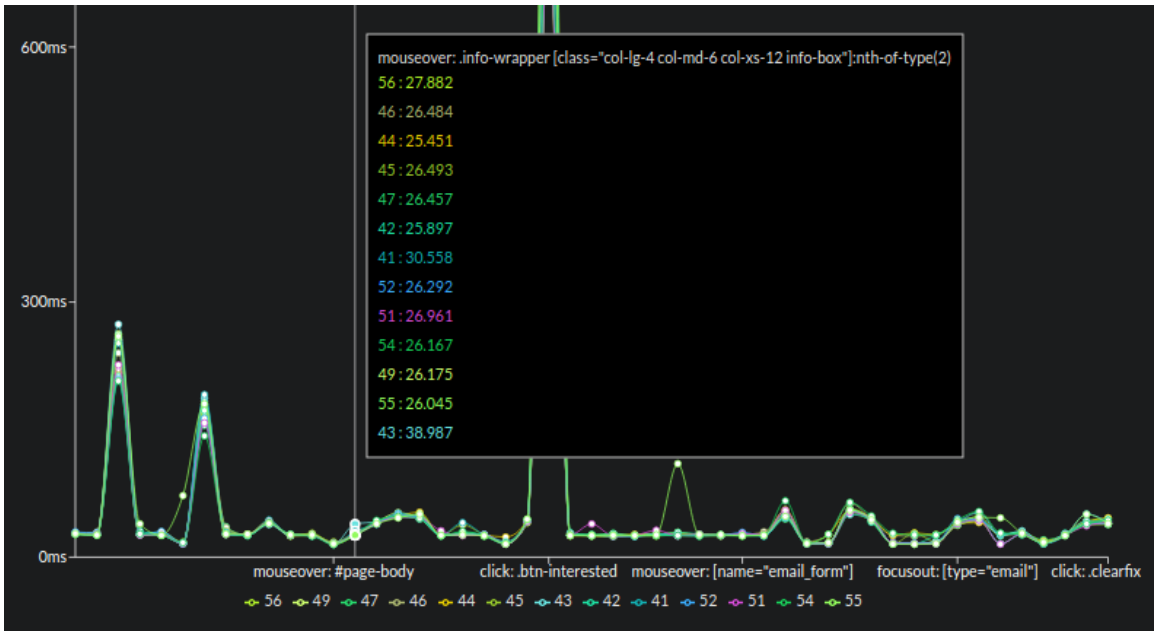


Obr. 5.13: Notifikácia o úspechu (vlastné)



Obr. 5.14: Notifikácia o chybe (vlastné)

Graf



Obr. 5.15: Graf vykonávania akcií (vlastné)

Jedna z hlavných funkcií aplikácií je testovanie výkonu stránky z pohľadu rýchlosti vykonania jednotlivých udalostí počas testovacieho procesu. Výsledky tohto typu testu sú zobrazené pomocou spojnicového grafu. Pre implementáciu grafu bola zvolená knižnica Recharts, ktorá má dobru integráciu s knižnicou React. Jednotlivé testy sú následne zobrazené ako jedna spojita čiara v grafe.

Pre zlepšenie použiteľnosti graf umožňuje zobraziť konkrétne časové hodnoty pre udalosti v jednotlivých testoch po prejdení myší nad bodmi reprezentujúcimi nejakú udalosť.

Porovnávač obrázkov

Porovnávač obrázkov zabezpečuje vizualizáciu výsledkov regresného testovania. Samotné porovnávanie obrázkov zabezpečuje knižnica react-image-diff, ktorá umožňuje 3 režimy porovnávania:

- **Rozdielové** – Zhodné pixeli sú zobrazené čiernou farbou, rozdielne sú následne zobrazené rozdielovou hodnotou medzi porovnávanými obrázkami.
- **Prechod** – Pomocou vstupu je možne meniť alfa kanál porovnávaného obrázku, kde je ako podklad použitý základný obrázok.
- **Oddeľovač** – Okno je rozdelené na ľavú a pravú časť, kde jedná časť predstavuje základný obrázok a druhá časť porovnávaný obrázok. Následne je možné pomocou vstupu meniť vertikálne pomer veľkosti jednotlivých častí okna.

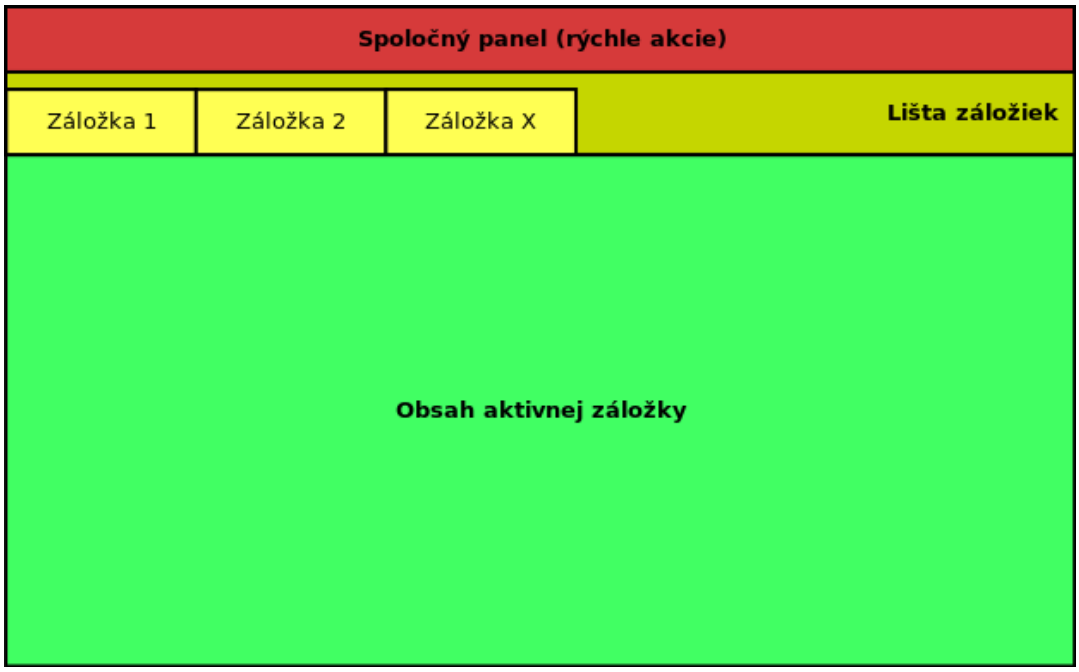
Obr. 5.16: Porovnávač obrázkov (vlastné)

Implementovaný porovnávač obrázkov bol rozšírený o ovládací panel, kde je možné voliť režim porovnávania a pre jednotlivé režimy následne upravovať hodnotu vstupu. Ovládací panel zároveň obsahuje aj vstup pre úpravu mierky zobrazovacieho okna porovnávača v rozsahu 25% až 150% pôvodnej veľkosti okna. Táto funkcionalita bola pridaná pre prípad, keď obrázky z testov sú vytvorené vo väčšom rozlíšení, ako má zobrazovacie zariadenie. Bez tejto funkcionality bolo možné sa cez obrázok len presúvať v horizontálnom a vertikálnom smere podľa veľkosti obrázka. Takéto riešenie, ale malo za následok ľahkú stratu celkovej informácie o chybe. Výhodou tohto riešenia je zároveň aj možnosť priblížiť obrázok a zamerať sa tak na menšie detaily.

Okrem samotného porovnávania obrázkov sa porovnávač obrázkov skladá aj z ľavého menu udalostí, kde kliknutím na udalosť dôjde k zmene obrázkov pre porovnávač obrázkov. Aby, bolo dosiahnutá čo najväčšia pracovná plocha pre samotný porovnávač obrázkov, sú jednotlivé udalosti reprezentované len poradovým číslom a ikonou udalosti. Dodatočné informácie o udalosti a výsledné skóre rozdielu obrázku oproti regresnému testu sa zobrazí vo vyskakujúcom okne, ktoré zapuzdruje jednotlivé položky v ľavom menu udalosti. Pre zlepšenie použiteľnosti bolo implementované aj farebné rozlíšenie obrázkov na základe skóre, kde zhodné obrázky aktuálneho testu vzhľadom na obrázky z regresného testu majú zelené pozadie a odlišné obrázky majú červené pozadie. Zároveň pre zvýraznenie aktívnej udalosti sa mení farba textu z bielej na čiernu a je pridané biele orámovanie.

5.5.3 Užívateľské rozhranie

Užívateľské rozhranie je navrhnuté ako jedná webová stránka. Tento koncept sa viac približuje klasickým počítačovým aplikáciam. Nevýhodou tohto riešenia je do značnej miery absencia používania tlačidla naspäť, ktoré môžu mať užívatelia vo zvyku používať. Použitím tohto tlačidla na jednostránkovej webovej stránke dôjde ale k uzavretiu celej stránky, bez ohľadu na počet vykonaných akcií a zmien obsahu. Preto bolo dôležité riešiť prepínanie obsahu ako súčasť webovej stránky a za týmto účelom bol základný koncept užívateľského rozhrania rozdelený na 3 horizontálne úrovne, ktoré sú znázornené nižšie na obrázku 5.17.



Obr. 5.17: Hlavné rozloženie prvkov UI (vlastné)

Spoločný panel

Je najvyššia lišta, ktorá ma slúžiť pre rýchle akcie, ktoré nie sú závislé na obsahu aktívnej záložky. V aktuálnom stave projektu obsahuje len tlačidlo pre vytvorenie novej aplikácie.

Lišta záložiek

Pomocou záložiek minimalizujem nefunkčnosť tlačidla naspäť. Implementácia záložiek je zároveň robustnejšia oproti klasickému tlačidlu naspäť, ktoré umožňuje sa vrátiť len o 1 stránku. Použitie lišty záložiek teda umožňuje ľahšie prepínať aktívny obsah a zároveň je týmto spôsobom ľahšie rozširovať ďalší obsah stránky.

Obsah aktívnej záložky

Hlavná časť užívateľského rozhrania, ktorá slúži ako kontajner pre zobrazenie jednotlivých častí obsahu webovej stránky, ktoré sú prístupne cez záložky.

5.5.4 Obsah webovej stránky

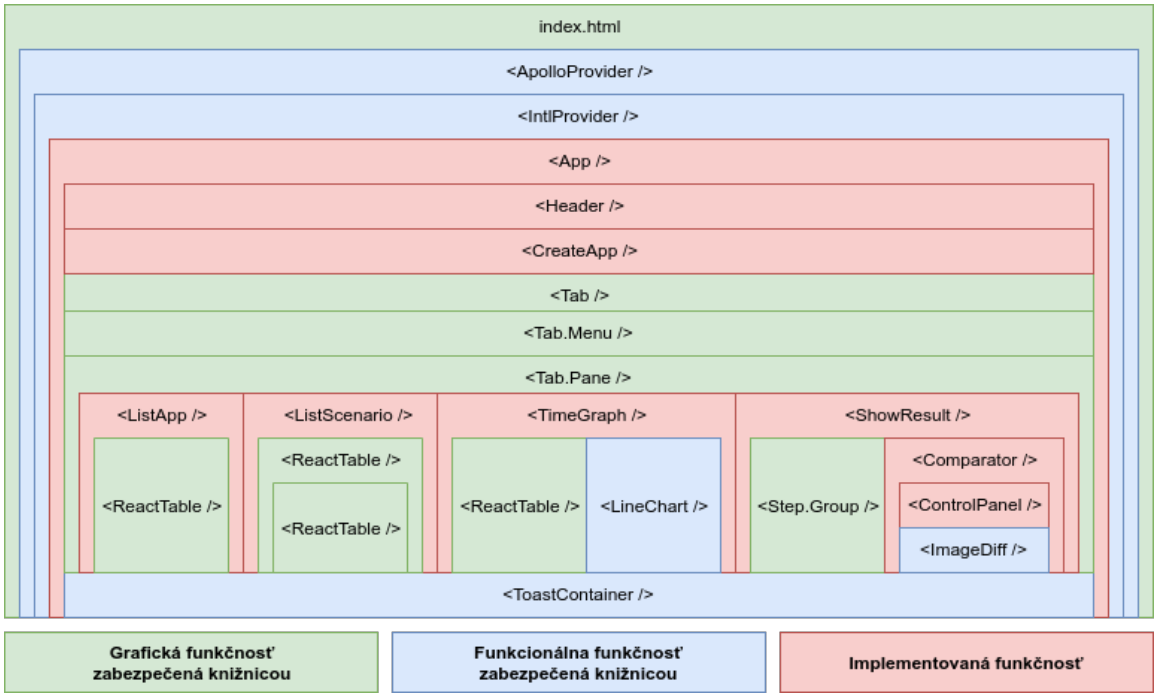
Hlavná časť obsahu webovej stránky je reprezentovaný obsahom samotných záložiek. Súčasťou jednotlivých zložiek sú grafické prvky popísané v sekcií 5.5.2 Prvky stránky. Aktuálna implementácia obsahuje štyri typy záložiek. Kde jednotlivé záložky majú pevne stanovený index a v prípade rovnakého indexu sa mení obsah danej záložky. Výsledný obsah s rozložením vyzerá nasledovne:

- **Zoznám aplikácií** [1] – Je postavený nad grafickým prvkom Zoznam a obsahuje zoznam všetkých zaregistrovaných aplikácií.
- **Zoznam scenárov a testov** [2] – Rovnako ako Zoznam aplikácií využíva prvok Zoznam a zobrazuje informácie o scenároch pre vybranú aplikáciu. Zároveň obsahuje po kliknutí ďalší prvok typu zoznam, kde sú zobrazené jednotlivé testy vybraného scenára.
- **Graf** [3] – Implementuje grafický prvok Graf (5.5.2) pre vybraný scenár.
- **Porovnávač** [3] – Implementuje grafický prvok Porovnávač obrázkov (5.5.2).

5.5.5 Programová štruktúra

Štruktúra webového rozhrania vychádzala hlavne na základe požiadaviek knižnice React, funkčných knižníc a grafického dizajnu webovej stránky. Na obrázku 5.18 je vidieť zjednodušenú hierarchickú implementáciu JSX elementov a prvkov vo webovom rozhraní. Pomocou farieb sú následne rozdelené jednotlivé prvky do týchto kategórií:

- **Grafická funkčnosť zabezpečená knižnicou** – Prvky implementujú prevažne grafické prvky webového rozhrania a ich hlavná časť je implementovaná pomocou externých knižníc a mierne upravená pre potreby grafického dizajnu.
- **Funkcionálna funkčnosť zabezpečená knižnicou** – Tieto prvky zabezpečujú určité funkcie samotného webového rozhrania, kde rovnako hlavná časť je implementovaná pomocou externých knižníc a mierna modifikovaná alebo rozšírená pre potreby webové rozhrania.
- **Implementovaná funkčnosť** – Vo väčšine prípadov sa jedná o zapuzdrenie iných externých prvkov pre lepšie prepojenie grafického a funkčného aspektu webové rozhrania.



Obr. 5.18: Zjednodušená štruktúra prvkov webového rozhrania (vlastné)

App

Element App vychádza z knižnice React, kde sa využíva ako základný element. Ako súčasť aplikácie je jeho hlavná funkcionalita primárne riadenie ostatných grafických prvkov. App generuje priamo element Tab a teda slúži ako centrálne miesto pre zmenu obsahu webového rozhrania.

Apollo

Apollo je knižnica pre komunikáciu pomocou protokolu graphQL a pre jej potreby je vložený JSX element ApolloProvider. Pre komunikáciu sa vo webovom rozhraní využívajú dva princípy:

- **withApollo** – Je funkcionalita, ktorá umožňuje priamy prístup k inštancií ApolloClient [16] pomocou referencie vytvorenej vo vlastnostiach priradeného JSX elementu a je použitý pre jednorázové volanie API na základe užívateľskej akcie. Pomocou priameho prístupu k ApolloClient umožňuje zasielať rôzne typy správ.
- **graphql** – Používa sa pre vytvorenie volaní API so zadefinovanou správou, ktorá sa už nemení. Rovnako, ako withApollo sú takto vytvorené objekty uložené vo vlastnostiach priradeného JSX elementu. Tento typ volaní API sa používa primárne pre zobrazovacie účely, kde je volanie API zavolané pri vytvorení (zobrazení) príslušného JSX elementu na stránke. Zároveň graphql umožňuje nastaviť automatické dotazovanie v pravidelných časových intervaloch, ktoré sa využíva pre zvýšenie interaktivity webového rozhrania s užívateľom.

Časový graf (TimeGraph)

Časový graf implementuje grafický prvok Graf a Zoznam popísaný v sekcií 5.5.2 Prvky stránky. Zoznam v tomto prípade slúži pre výber testov, ktorých časové údaje o dobe vykonávania jednotlivých udalostí sa následne zobrazia v príslušnom grafe.

Aby sa minimalizoval počet volaní pri každom zrušení a obnovení zobrazenia rovnakého testu, časový graf si ako súčasť interných štruktúr udržiava načítané hodnoty. Ak bol test niekedy zobrazený, nedochádza k volania API rozhrania s požiadavkou o výsledky testu, ale sú načítané dáta z interných štruktúr.

Grafické prevedenie čiar na grafe je generované náhodne v hexadecimálnom formáte pomocou funkcie:

```
1 '#'+(Math.random()*0xFFFFFF<<0).toString(16)
```

Výpis 5.2: Generovanie farby spojníc (prevzaté [4])

Takéto riešenie, ale značne zhoršovalo čitateľnosť grafu v prípade, že došlo k vygenerovaniu tmavých farieb. Preto je výsledok generovania cyklický a kontrolovaný pomocou knižnice tinycolor2 [12], pokiaľ jas vygenerovanej farby je tmavý.

Zobrazenie výsledkov (ShowResult)

Využíva rovnaký princíp ako časový graf, kde na ľavej strane je zobrazený zoznam udalosti implementovaný pomocou JSX elementu Step.Group z knižnice smenatic-ui-react. Po kliknutí na nejaký element sa na pravej časti stránky zobrazí grafický prvok Porovnávač obrázkov s nastavenými obrázkami pre vybranú udalosť. Jednotlivé elementy sú pre lepšiu použiteľnosť farebne rozlíšene na základe skóre z SSIM metódy, kde hodnoty reprezentujú:

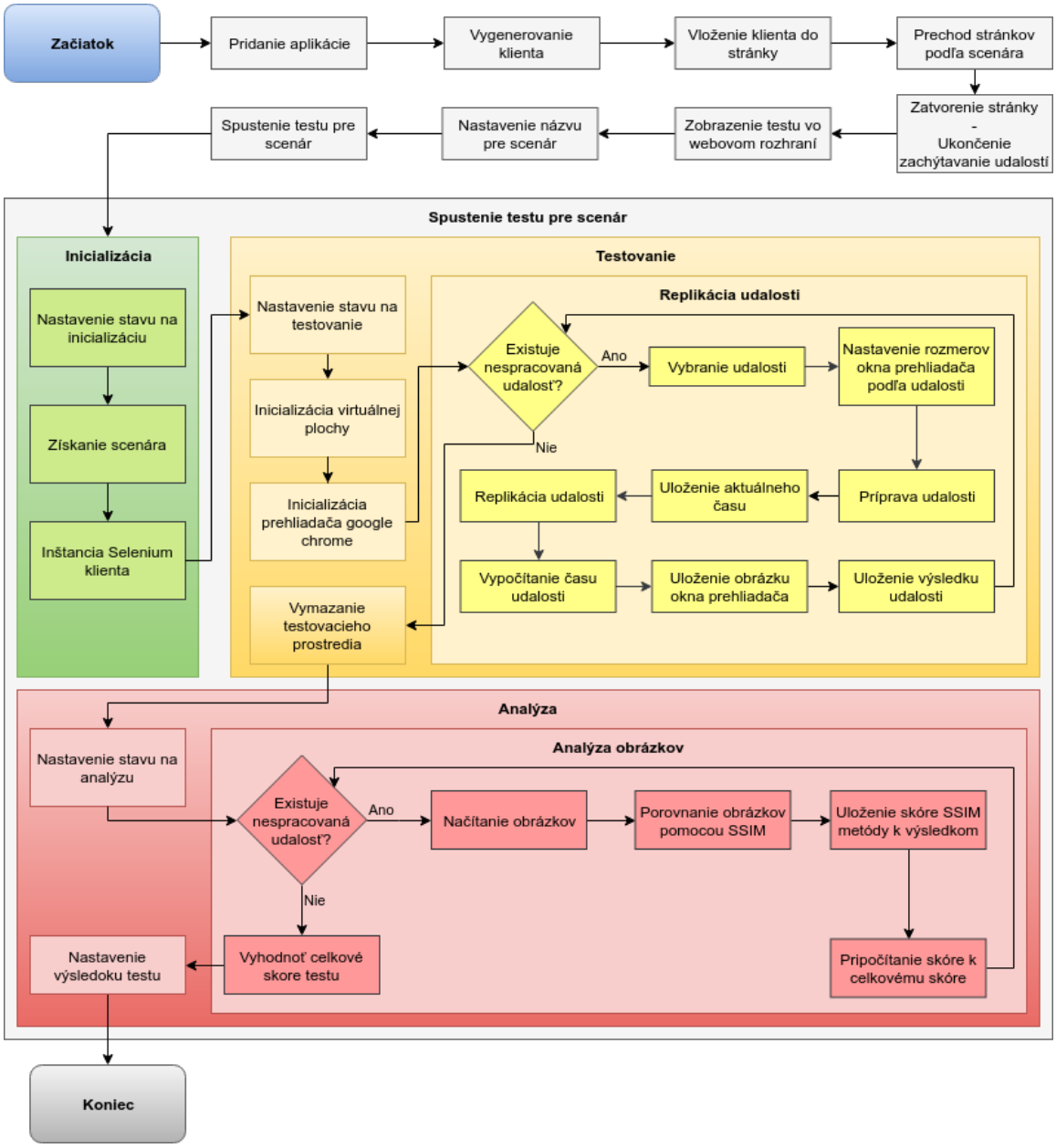
- **1.0** – Obrázky sú zhodné a udalosť má nastavené zelené pozadie.
- **<1.0** – Obrázky sú rozdielne a udalosť má červené pozadie.

5.6 Testovací proces

Všetky prvky aplikácie by mali spoločne umožňovať jednoduché testovanie webových aplikácií. Základnú myšlienku testovacieho procesu webových aplikácií môžeme vyjadriť jednoduchým blokovým diagramom, kde jednotlivé bloky abstrahujú určitú funkcionalitu aplikácie. Diagram na obrázku 5.19 je rozdelený na dve hlavné častí:

- Príprava testu – Popisuje úkony potrebné pre vytvorenie scenára.
- Testovacia časť – Popisuje ako prebieha testovací proces.

Testovacia časť je následne rozdelená podľa podporovaných stavov testov na 3 sekcie, ktoré sú popísané v sekcií 5.1 Komunikácia v časti 5.1.1 Stav. A ktoré reprezentujú určitú triedu z častí 5.3.3 Riadenie testov a 5.3.3 Testovanie zo sekcie 5.3 Server.



Obr. 5.19: Zjednodušený blokový diagram testovacieho procesu (vlastné)

Kapitola 6

Testovanie

Testovanie aplikácie prebiehalo na reálnych produktoch za pomoci spolupráce so spoločnosťou inQool, ktorá umožnila využiť dve vyvíjané webové aplikácie. Tieto aplikácie patria do kategórie elektronických obchodo,v nakoľko pomáhajú pri vytvorení obchodnej transakcie. Webové aplikácie zároveň reprezentujú iný typ webovej stránky, čo umožnilo lepšie otestovať správanie testovacej aplikácie. Jednotlivé testovacie aplikácie sú reprezentované pomocou vývojových verzií webových stránok:

- **Kreativátor** (www.kreativator.eu) – Dynamická webová stránka s veľkým množstvom interaktívnych prvkov. Webová stránka slúži pre objednanie fyzických produktov s podporou košíka.
- **iQ metropolis** (www.iqmetropolis.cz) – Prevažne statická jednoduchá webová stránka s malým množstvom prvkom, slúžiaca pre objednávku dodatočných informácií o poskytnutí služieb.

Testy aplikácie boli následne rozdelené na 3 typy, kde sa skúmali rôzne aspekty testovacej aplikácie a kde ako súčasť 2 typov testov boli použité vývojové verzie webových stránok poskytnuté spoločnosťou inQool. Jednotlivé typy testov sú popísane v sekciách 6.1 Grafický dizajn až 6.3 Spoľahlivosť.

6.1 Grafický dizajn

Cieľom tohto testu bolo overiť stav a ďalší smer vývoja webové rozhrania a to ako po funkčnej, tak aj vizuálnej stránke, aby čo najviac vyhovoval koncovému používateľovi.

Súčasťou tohto testu bolo oslovenie externých spoločností, ktoré ako súčasť vývoja programujú aj webové aplikácie. Počas stretnutia im bolo ukázané webové rozhranie aplikácie s pripraveným prostredím a vytvorenými scenármi s vykonanými testami. Následne boli požiadaní, aby si webové rozhranie prezreli a otestovali jednotlivú funkcionalitu. V prípade nejasnosti im bola poskytnutá pomoc. Osoba, ktorá testovala webové rozhranie, bola požiadaná o vyplnenie jednoduchého dotazníka. Pre potreby dotazníka sa používali prevažne stupnice, kde najnižšie číslo reprezentovalo najhorší prípad a najvyššie číslo ideálny stav. Otázky vo formulári boli nasledovné:

- **Príde Vám webové rozhranie jednoduché?** (Stupnica: 1 - 4) – Cieľom bolo overiť nedostatok alebo nadbytočnosť ovládacích prvkov a zároveň jednoduchosť celého webové rozhrania.

- **Koľko času Vám zabralo pochopenie rozhrania?** (Doba trvania: minúty) – Doplnková otázka pre overenie jednoduchosti webového rozhrania.
- **Príde Vám grafické spracovanie ucelené?** (Stupnica: 1 - 4) – Overenie grafického dizajnu po vizuálnej stránke.
- **Vyhovoval Vám tmavý dizajn webovej stránky?** (Stupnica: 1 - 5) – Zistenie prijateľnosti tmavého dizajnu stránky, ktorý je na webových stránkach menej bežný.
- **Mali ste problémy, že aplikácia je jedna webová stránka?** (Možnosti: Áno - Nie) – Overenie ovládania webového rozhrania.
- **Myslíte si, že je program použiteľný?** (Stupnica: 1 - 4) – Možnosť použitia aplikácie ako súčasť vývojového cyklu.
- **Pochopili ste z webového rozhrania aký je zmysel programu?** (Stupnica: 1 - 4) – Celkové hodnotenie webového rozhrania s ohľadom na funkcionalitu aplikácie.
- **Vyskytli sa počas testovania nejaké závažné problémy?** (Stupnica: 1 - 4) – Prieskum funkčnosti aplikácie.
- **Iné pripomienky?** (Text) – Dodatočné informácie alebo podnety od testovacej osoby, mimo testovacích otázok.

6.1.1 Výsledky

Výsledky tohto testu sú ale značne nedostatočné, nakoľko sa do testovania grafického dizajnu zapojili len tri spoločnosti, ktoré poskytli len jedneho testera. Na určité aspekty výsledkov je teda vhodné sa pozerať len ako na informatívne. Výsledné skóre pre jednotlivé otázky bolo nasledujúce:

- **Príde Vám webové rozhranie jednoduché?** – **Priemerné skóre: 3** zo 4 – Webové rozhranie môžeme považovať za jednoduché ale stále je priestor na optimalizáciu.
- **Koľko času Vám zabralo pochopenie rozhrania?** – **Najväčšia hodnota: 5 minút** – Najhorší časoví interval 5 minút považujem za veľmi dobrý výsledok a potvrdzuje predchádzajúcu otázku.
- **Príde Vám grafické spracovanie ucelené?** – **Priemerné skóre: 3.33** zo 4 – Ukážková verzia mala určité kozmetické nedostatky v chýbajúcich nápovedách a nejednotnom štýle tlačidiel.
- **Vyhovoval Vám tmavý dizajn webovej stránky?** – **Priemerné skóre: 4** z 5 – Tmavý dizajn bol prijatý kladne a nie je teda dôvod ho meniť na svetlý.
- **Mali ste problémy, že aplikácia je jedna webová stránka?** – **Jednotná odpoveď: Nie** – Obavy s problémom používania tlačidla na predchádzajúcu stránku sa nepotvrdili a užívatelia sú na daný štýl stránok najskôr zvyknutý.
- **Myslíte si, že je program použiteľný?** – **Priemerné skóre: 3.33** zo 4 – kladné skóre môžeme chápať, že testovaciu aplikáciu, by bolo možné uplatniť vo vývojom cykle.

- **Pochopili ste z webového rozhrania aký je zmysel programu? – Priemerné skóre: 3.33** zo 4 – Užívateľské rozhranie odpovedá požadovanej funkcionalite. Z pozorovania pri testovaní ale usudzujem, že porovnávač obrázkov dostatočne neodráža, že sa jedná o funkcionalitu regresného testovania.
- **Vyskytli sa počas testovania nejaké závažné problémy? – Priemerné skóre: 1.33** zo 4 – Táto otázka je značne sporná nakoľko ukážka prebiehala na pripravenom prostredí. Z webového rozhrania sa ale počas testovania nevyskytli žiadne závažné problémy, okrem nejasných chybových hlásení, ktoré nemali vplyv na funkčnosť.
- **Iné pripomienky?** – Neboli žiadne pripomienky.

6.2 Tvorba scenárov

Test sa zameriava na splnenie formálneho zadania diplomovej práce pre vytvorenie testovacieho scenára. V tomto teste boli použité poskytnuté webové aplikácie a pomocou aplikácie bola snaha vytvoriť jednotlivé prípady testovacieho scenára, ktoré sú uvedené v kapitole 4 Návrh a sekcií 4.1 Testovací scenár. Ako súčasť tvorby testovacieho scenára bolo potrebné vziať do úvahy aj funkčnosť webových aplikácií, kde chýbajúca funkčnosť neumožňovala tvorbu niektorých prípadov testov.

6.2.1 Výsledky

Pre overenie výsledkov sa použil video záznam z tvorby jednotlivých scenárov kombinácií s prvým spustením testu vytvoreného scenára. Jednotlivé scenáre testov odpovedajú len približne testovacím scenárom a sú vytvorené podľa možnosti testovanej webovej aplikácie.

Počas testovania bol na webovej stránke kreativator problém s nefunkčnou tvorbou objednávky s možnosťou zasielania noviniek. Tento problém neumožňoval úspešne ukončiť objednávku, ale pre overenie funkčnosti prototypu testovacej aplikácie, presmerovanie na inú stránku nemá veľký vplyv. Jednotlivé scenáre sú na nasledujúcich videách zverejnených na portály YouTube alebo uvedené na médiu ako príloha:

- **Prihlásenie užívateľa**
  - Kreativator
- **Tvorba objednávky jedného produktu**
  - Kreativator
  - iQ metropolis
- **Tvorba objednávky s vyhľadávaním**
  - Kreativator

Celkovo sa podarilo s možnosťami testovaných webových stránok zrealizovať 3 zo 4 scenárov. Scenár Zrušenie objednávky nebolo možné vytvoriť nakoľko ani jedná testovacia stránka neobsahovala túto funkcionalitu. Zároveň testovacie stránky pre potreby vytvorenia scenára potrebujú mať vytvorené presmerovanie dotazov na serverovú časť aplikácie, alebo povolenú výnimku pre zasielanie dotazov na iný server a preto je testovacia vzorka webových aplikácií tak malá. Nakoľko v testoch sú zahrnuté často používané ovládacie prvky ako je

klikanie na odkazy, vyplňovanie formulárov a prechod webovou stránkou, môžeme tento test považovať za úspešný a test dokazuje že pomocou testovacej aplikácie je možné vytvárať testovacie scenáre.

Test zároveň môžeme považovať aj za úspešný z pohľadu nájdenia chyby, kde počas zrušenia odoberania noviniek počas vytvárania objednávky na testovanej stránke kreativator dochádza k dvojitému kliknutiu z dôvodu prekrývajúcich sa elementov. V tomto prípade je problém spôsobený, že pri testoch nie je simulovaný úplne rovnaký pohyb myši ako počas tvorby testu a poloha myši je nastavená na polohu elementu pomocou selektora a knižnice selenium. To môže viesť na rozdielne chovanie počas testu a náročnejšiu tvorbu testovacích scenárov.

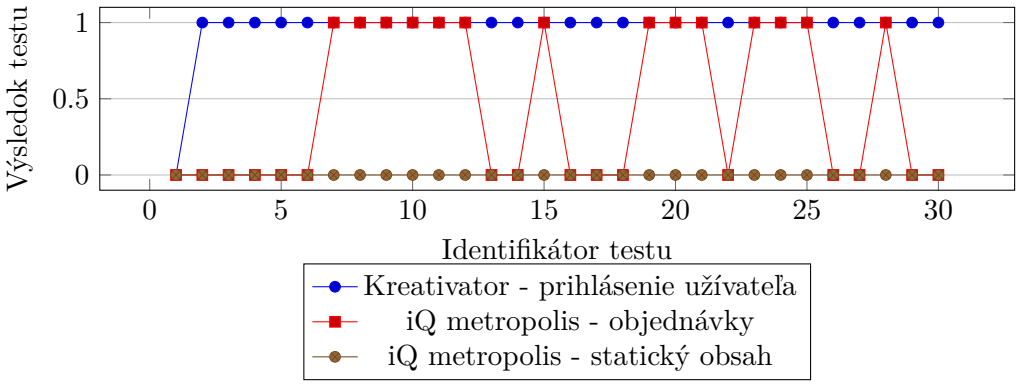
6.3 Spoľahlivosť

Účelom spoľahlivosti bolo otestovať na vybraných scenároch celkovú funkčnosť aplikácie a výsledky samotného testovacieho procesu z pohľadu regresného testovania. V tomto prípade testovanie prebehne pomocou serie 30 po sebe idúcich testov, ktoré neovplyvňujú stav testovanej aplikácie a bude sa overovať zastúpenie falošných poplachov.

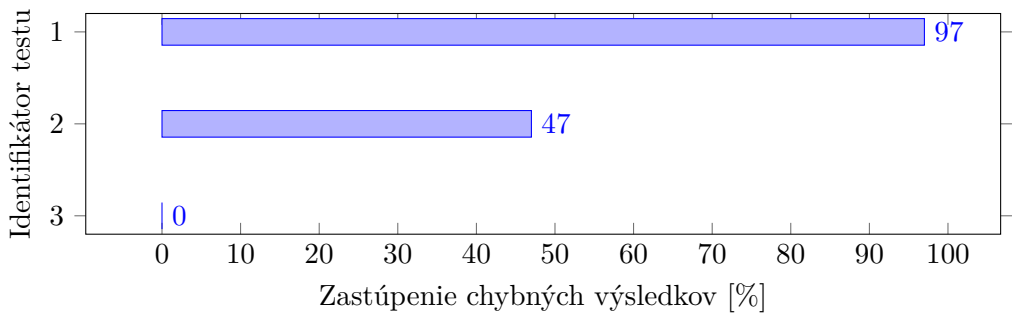
Pre tieto testy boli použité existujúce testy vytvorené v sekcií 6.2 Tvorba scenárov. Testy boli zvolené tak, aby reprezentovali rôznu náročnosť s ohľadom na samotný obsah, ktorý sa ako súčasť testu vyhodnocoval vo forme obrázkov z okna prehliadača. Nasledujúce testy sú zároveň aj číselne označené pre potreby grafov:

1. **Kreativator – prihlásenie užívateľa** – Zložitý test s dynamickými prvkami a formulármi.
2. **iQ metropolis – objednávka** – Jednoduchšia stránka s formulármi a malým množstvom dynamického obsahu.
3. **iQ metropolis – statický obsah** – Dodatočne vytvorený test s prezeraním webovej stránky bez dynamického obsahu.

6.3.1 Výsledky



Obr. 6.1: Výsledky testov spoľahlivosti (vlastné)



Obr. 6.2: Chybné výsledky testov spoľahlivosti (vlastné)

Na základe grafov na obrázkoch 6.1 Výsledky testov spoľahlivosti a 6.2 Chybné výsledky testov spoľahlivosti, môžeme usúdiť, že testovací nástroj generuje viac falošných poplachov, čím viac dynamického obsahuje existuje na stránke. Tento problém je spôsobený viacerými faktormi od rýchlosti načítania skriptov, obrázkov a iného obsahu, tak aj zachytávanie obrázku okna prehliadača nie je vždy vykonané v rovnakom čase ako pri regresnom teste. To vedie k značnej neurčitosti obsahu okna prehliadača, čo má za následok rozdiel v obrázkoch. Zároveň, aby bol test označený ako neúspešný stačí, že je detekovaný rozdiel v jednom obrázku a to na úrovni 1 pixelu (záleží od veľkosti obrázka).

Z analýzy obrázkov vytvorených počas testov, ktoré sú uvedené na médiu v prílohách, môžeme usúdiť, že testovací proces vždy korektne zreplikoval udalosti v scenári a skončil na korektnej poslednej stránke. Táto funkcionalita testovacieho nástroja tiež ale nefunguje spoľahlivo na zložitých webových stránkach. Ako bolo uvedené v sekcií 6.2 v časti 6.2.1 Výsledky, môže mať replikácia udalosti kliknutia problém s prekrývajúcimi sa elementami. Zároveň pre výber elementov sa používajú CSS selektory, ktoré môžu v prípade dynamickej stránky byť nefunkčné.

6.4 Zhrnutie testovania

Testy poukázali na niekoľko nedostatkov v implementácií, ktoré sa prejavujú prevažné na zložitých stránkach s veľkým množstvom dynamického obsahu. Celkovo má testovanie webových aplikácií problém s dynamickým obsahom, čo je spôsobené samotným spôsobom testovania. Tento problém by mohol byť jednoducho maskovaný možnosťou nastavenia tolerancie pre jednotlivé udalostí v scenári. Takéto riešenie, by ale značne znížilo spoľahlivosť testov, nakoľko skóre sa počíta pre celkový obrázok okna prehliadača a ľahko by došlo k ignorovaniu chyby. Najvhodnejším riešením tohto nedostatku vidím v dvoch možnostiach rozšírenia aplikácie:

- **Testovacie zóny** – Nastavenie testovacích zón v jednotlivých obrázkoch. To by viedlo ale k zložitejšiemu procesu návrhu testovacieho scenára.
- **Dočasný video záznam** – Nakoľko prehliadač beží vo virtuálnom displeji, bolo by možné zachytávanie obrázkov nahradiť vytvorením videa, z ktorého by sa následne extrahovali požadované obrázky. V tomto prípade treba ale počítať, že čas testovacieho procesu a požiadavky na pamäťové zdroje budu väčšie a môžeme predpokladať, že požiadavky na čas testovacieho procesu budú exponenciálne rásť s veľkosťou testovacieho scenára a zložitosťou testovanej webovej stránky.

Zároveň testovacia aplikácia je značne závislá v aktuálnom stave na samotnej testovanej webovej stránke, kde vytváranie korektných scenárov pre niektoré webové stránky môže byť značné náročne a neefektívne. To platí prevažne na webové stránky s veľkým počtom prvkov, kde počet prvkov presahuje hranicu 200. Je ale potrebné vziať aj do úvahy štruktúru testovanej webovej stránky a veľkosť samotného testovacieho scenára.

V základe aby bolo možné simulovať správny chod práce na webovej stránke je zachytávaný prechod nad všetkými prvkami. Výsledkom je veľký počet udalostí, ktoré je potrebné simulovať a pre ktoré sa generujú ako súčasť testovacieho procesu obrázky. Tieto obrázky následne treba aj analyzovať, čo zaberie veľké množstvo strojového času. Hlavný problém ale je, že väčšina takýchto udalostí môže byť pre potreby samotného testu nepotrebná. Preto by bolo vhodné optimalizovať aj spôsob zachytávania udalostí na strane klienta, alebo umožniť pomocou webového rozhrania odstrániť nepotrebné udalostí ako súčasť scenára.

Jednoduchý návrh grafického webového rozhrania aplikácie s malým množstvom prvkov sa na základe dotazníka zdal byť ako dobré rozhodnutie. Musím ale brať do úvahy mali počet ľudí, ktorý produkt testovali. Nakoľko ale väčšina z nich má skúsenosti s vývojom webových aplikácií a niektorí patria do cieľovej skupiny užívateľov, ich odozva je vhodná pre určenie ďalšieho vývoja grafického dizajnu, ktoré by sa mohlo ešte zjednodušiť a lepšie upraviť pre oznámenie chýb. Na základe dotazníka a osobných rozhovor bolo by vhodné zapracovať nasledujúce zmeny:

- Zrušenie horného panela.
- Nastaviť vlastné chybové správy.
- Zrušiť prepisovanie záložiek pre porovnávanie obrázkov a graf.
- Pridať ochranu pre akciu naspäť v prehliadači.
- Definovať nadpis pre stĺpec s nastavením regresného testu.

Kapitola 7

Záver

Cieľom bolo vytvoriť testovaciu aplikáciu, ktorá by umožňovala ľahko vytvárať testovacie scenáre a automatizovať ich spúšťanie. A to hlavne so zameraním na elektronické obchody. Výsledkom práce je prototyp testovacej aplikácie pre testovanie webových stránok, kde sa k jednotlivým testom pristupuje z pohľadu čiernej skrinky. Aplikácia umožňuje regresné testovanie proti staršej verzii testu a zároveň implementuje výkonnostné testovanie založené na čase potrebnom pre replikáciu udalosti. Celá aplikácia je navrhnutá z troch hlavných častí, ktoré sú tiež modulárne:

- **Server**
- **Klient**
- **Webové rozhranie**

Vďaka klientovi, ktorý môže byť vložený do webovej stránky a spracováva udalosti, ktoré sú následne uložené na servery, ako kroky testovacieho scenára, je vytváranie scenárov značne jednoduché. Zároveň takéto riešenie umožňuje v budúcnosti vytvárať užívateľské scenáre, ktoré môžu byť zložené z viacerých užívateľských scenárov. A tým overovať napríklad najviac používanú funkčnosť, alebo obsah webovej stránky.

Pomocou komunikačného rozhraniu graphQL, ktoré je medzi jednotlivými časťami testovacej aplikácie je možné veľmi ľahko rozširovať aplikáciu o podporu ďalších aplikácií. Zároveň je možné takto integrovať aplikáciu do integrovaného vývojového prostredia, alebo správcov zdrojových kódov, ako súčasť procesu priebežnej integrácie (angl. Continuous Integration). K integrácií pre spúšťanie testov postačuje, aby bolo možné na server zaslať príslušné HTTP požiadavky, čo dnes umožňuje väčšina nástrojov napríklad pomocou aplikácie alebo knižnice curl.

Pre overenie funkčnosti aplikácie sa použili testovacie scenáre uvedené v kapitole 4 Návrh, sekcia 4.1 Testovací scenár. Zo všetkých 4 scenárov sa úspešne podarilo vytvoriť a otestovať 3 testovacie scenáre. Testy prebiehali na skutočných webových aplikáciach, ktoré sa zameriavajú na vytvorenie obchodnej transakcie. Posledný scenár nebolo možné vytvoriť z dôvodu chýbajúcej funkcionality v testovaných webových aplikáciach. Ale predchádzajúce 3 scenáre obsahovali všetky bežné ovládacie prvky, ktoré sa vyskytujú na webových stránkach a preto by nemal byť problém v prípade možnosti vytvoriť aj 4. scenár.

Na základe testov aplikácie boli zistené niektoré nedostatky testovacej aplikácie, ktoré sa prejavujú hlavne na stránkach s veľkým počtom prvkov a to hlavne dynamických. Kde dochádzalo k veľkému počtu falošných výsledkov a úspešnosť testov bola vo výsledku 0%. Testovacia aplikácia je teda v aktuálnom stave vhodná prevažne na testovanie jednoduchších

webových stránok, ktoré sa zameriavajú na vytvorenie obchodných transakcie. Aplikáciu ale je možné využiť pre ľubovoľnú webovú aplikáciu.

Z pohľadu agilného vývoja je aplikácia vhodná až do neskorších iterácií, keď existuje už funkčný prototyp, na ktorom je možné vytvoriť testovací scenár. Preto aplikácia nie je vhodná pre agilnú metodiku TDD, ktorá vyžaduje testy pred zahájením vývoja. Inak je možné testovaciu aplikáciu vďaka podpore viacerých scenárov pre testovanú aplikáciu využiť všade, kde existujú z časti funkčné webové stránky pre regresné testovanie alebo A-B testy.

Využiť testovaciu aplikáciu ako súčasť agilného vývoja je tiež vhodné z dôvodu centrálnej správy testov a testovacích scenárov, ktoré sú prístupné cez webové rozhranie, čo poskytuje nasledujúce výhody:

- **Komunikácia** – V prípade komunikácie medzi jednotlivými členami tímu pri riešení problémov.
- **Jednotné testovacie prostredie** – Testovanie prebieha vždy na tom istom servery a prostredie je teda stabilné a nemalo by ovplyvniť výsledky testov.

Tieto výhody sú hlavne oproti jednoduchým testovacím nástrojov, ktoré bežia lokálne, ako napríklad súčasť integrovaného vývojového prostredia.

Literatúra

[1] *FURPS*. Wikimedia Foundation, Inc., [Online; navštívené 10.11.2017].
URL https://en.wikipedia.org/wiki/FURPS

[2] *SSIM*. Wikimedia Commons, [Online; navštívené 05.03.2018].
URL https://cs.wikipedia.org/wiki/SSIM

[3] Ammann, P.; Offutt, J.: *Introduction to Software Testing*. Cambridge University Press, 2008, ISBN 978-0-521-88038-1.

[4] Azimi, M.: *Random color generator*. Stack Exchange Inc, [Online; navštívené 29.03.2018].
URL https://stackoverflow.com/questions/1484506/random-color-generator

[5] Berners-Lee, T.: *WorldWideWeb: Proposal for a HyperText Project*. CERN, 1989, [Online; navštívené 03.12.2017].
URL http://info.cern.ch/hypertext/WWW/Proposal.html

[6] Berners-Lee, T.; Fielding, R.; Masinter, L.: *Uniform Resource Identifier (URI): Generic Syntax*. Internet Engineering Task Force, 2005, [Online; navštívené 20.10.2017].
URL https://www.ietf.org/rfc/rfc3986.txt

[7] Berners-Lee, T.; Masinter, L.; McCahill, M.: *Uniform Resource Locators (URL)*. Internet Engineering Task Force, 1994, [Online; navštívené 20.10.2017].
URL https://www.ietf.org/rfc/rfc1738.txt

[8] Crispin, L.; Gregory, J.: *Agile Testing: A Practical Guide for Testers and Agile Teams*. Pearson Eduction, Inc., 2009, ISBN 978-0-321-53446-0.

[9] Doglio, F.: *Pro REST API Development with Node.js*. Apress, 2015, ISBN 978-1-4842-0918-9.

[10] Ecma International: *ECMAScript® 2015 Language Specification*. Ecma International, 2015, [Online; navštívené 27.11.2017].
URL http://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf

[11] Fielding, R.; Gettys, J.; Mogul, J.; aj.: *Hypertext Transfer Protocol – HTTP/1.1*. Internet Engineering Task Force, 1999, [Online; navštívené 20.10.2017].
URL https://www.ietf.org/rfc/rfc2616.txt

[12] Grinstead, B.: *TinyColor*. GitHub, Inc., [Online; navštívené 29.03.2018].
URL http://bgrins.github.io/TinyColor/

[13] Kieffer, R.; Stein, A.; Oldridge, P.: *uuid*. GitHub, Inc., [Online; navštívené 15.03.2018].
URL https://github.com/kelektiv/node-uuid/blob/HEAD/README_js.md

[14] McConnell, S.: *Dokonalý kód*. Computer Press, a.s., 2005, ISBN 80-251-0849-X.

[15] Pixley, T.: *1. Document Object Model Events*. W3C, 2000, [Online; navštívené 20.03.2018].
URL https://www.w3.org/TR/DOM-Level-2-Events/events.html

[16] Rayzis, P.; Ferrari, V.: *withApollo*. GitHub, Inc., [Online; navštívené 10.03.2018].
URL https://www.apollographql.com/docs/react/api/react-apollo.html#withApollo

[17] Rostecký, J.: *B2B, B2C, B2G a další zapeklité značky, které neznačí nic, ale vlastně všechno...*. Jiří Rostecký, [Online; navštívené 25.11.2017].
URL https://mladypodnikatel.cz/b2b-b2c-b2g-c2b-b2a-b2e-b2r-c2c-c2g-g2b-g2c-g2g-t950

[18] Sustrik, M.; Lucina, M.: *zmq_inproc(7)*. iMatix Corporation, [Online; navštívené 28.01.2018].
URL http://api.zeromq.org/2-1:zmq-inproc

[19] W3C Groups: *HTML & CSS*. W3C, [Online; navštívené 21.10.2017].
URL https://www.w3.org/standards/webdesign/htmlcss

[20] W3C Groups: *HTML 4.01 Specification*. W3C, 1999, [Online; navštívené 21.10.2017].
URL https://www.w3.org/TR/html401/cover.html

Príloha A

Obsah CD

Pribalené CD obsahuje všetky zdrojové kódy v ktorých prebiehala implementácia a doplnkové materiáli.

- **src/** – Zdrojové kódy aplikácie.
  - **backend** – Zdrojové kódy servera.
  - **client** – Zdrojové kódy klienta
  - **frontend** – Zdrojové kódy webového rozhrania.
- **pdf/** – PDF verzia diplomovej práce.
- **tex/** – Zdrojové súbory pro LATEX k vytvoreniu PDF verzie diplomovej práce.
  - **tex/obrazky-figures** – Použité obrázky v diplomovej práci.
- **obrazky-pracovne/** – Pracovné verzie obrázkov použitých v práci pre webovú aplikáciu draw.io.
- **testy/** – Vytvorené materiály ako súčasť testovania aplikácie.
  - **databaza/** – Záloha databázy a exporty riadiacich dokumentov z testovania aplikácie.
  - **dokumenty/** – Výsledky dotazníka.
  - **obrazky/** – Obrázky okna prehliadača z testovania aplikácie.
  - **videa/** – Video záznamy z testovania tvorby scenárov.